Understanding Undo Support in Cocoa

MacTech Magazine
April • 2008

MACTECH®

The Journal of Macintosh Technology

RubyCocoa

A new way to write Cocoa applications

Modular DMG deployment

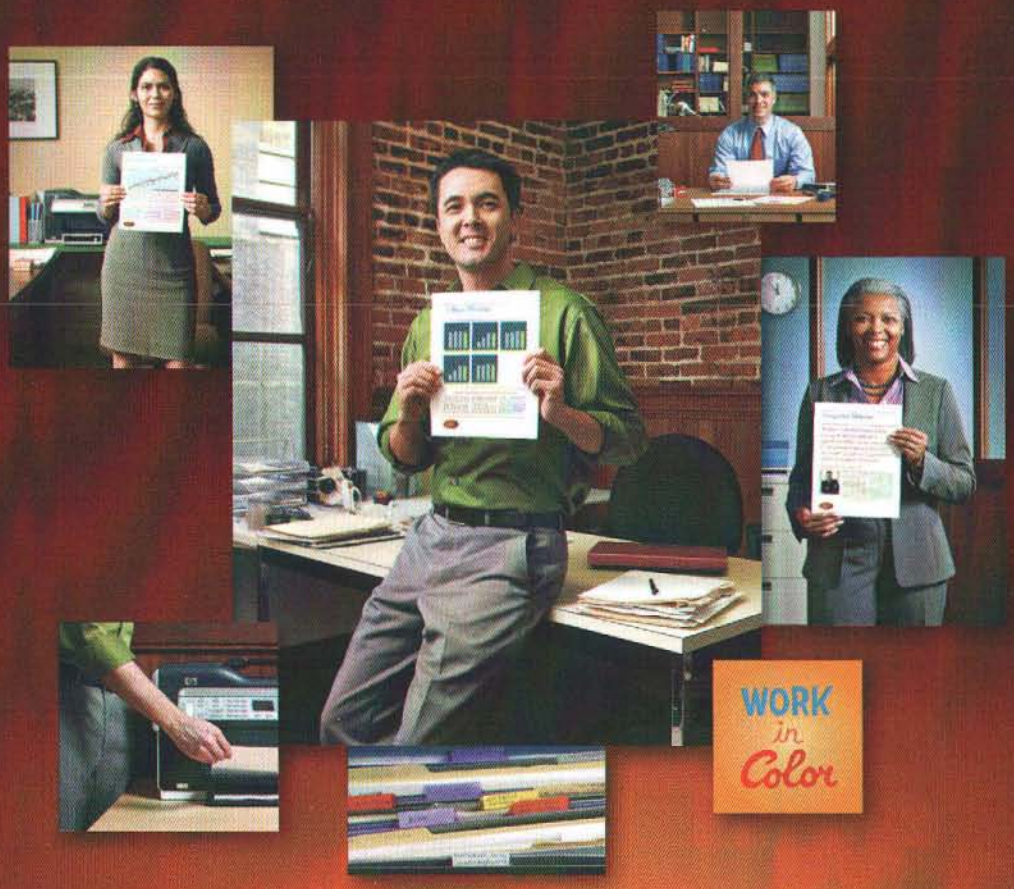
Test-Driven Development Using AppleScript

Data Manipulation with PHP

MACTECH.COM

$8.95 US, $12.95 Canada

ISSN 1067-8360 Printed in U.S.A.



UP TO 25% LOWER COST PER PAGE THAN LASER PRINTERS.*
HP OFFICEJET PRO ALL-IN-ONES STARTING AT $299.

They fax, scan, copy and deliver laser-quality printing—at an impressively low cost per page. Individual ink cartridges save money too, because you only replace what runs out. HP, *PC Magazine*'s Readers' Choice for service and reliability for 15 years running.

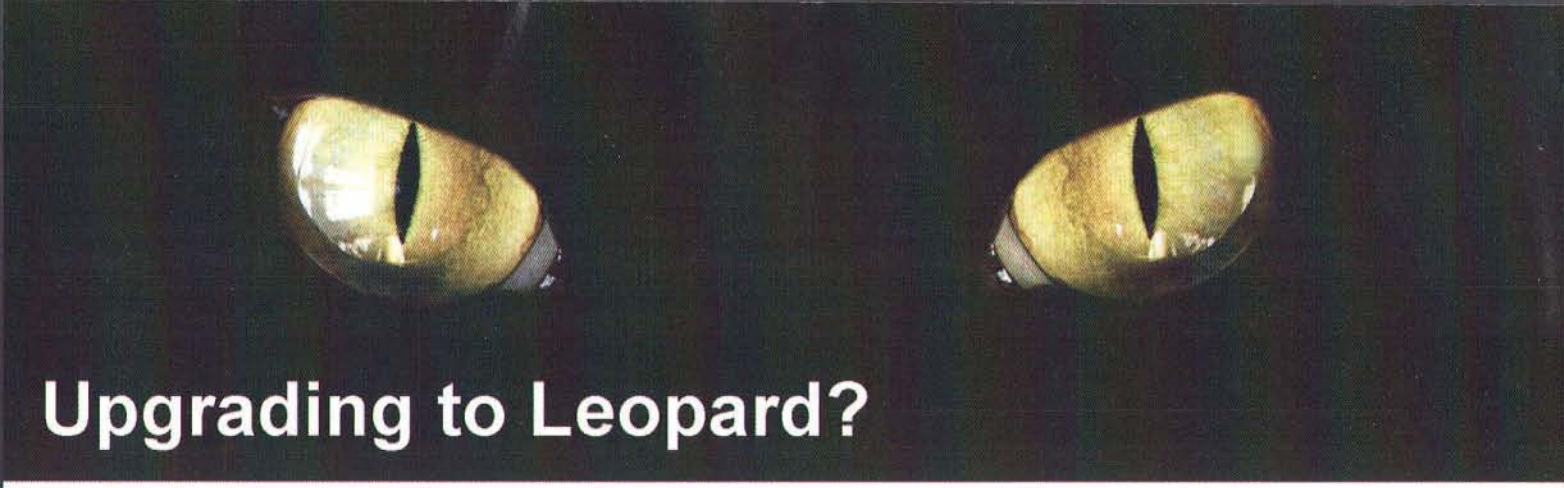
BEST
HP Officejet Pro L7780 AiO.
Plus wireless networking.
$499.

BETTER
HP Officejet Pro L7680 AiO.
Plus two-sided printing.
$399.

GOOD
HP Officejet Pro L7580 AiO.
Up to 35 pages a minute.
$299.

To learn more, visit hp.com/go/doesitall. Call 1-800-888-3119. Or visit your local reseller or retailer.

*Cost per page (CPP) laser supplies comparisons based on published manufacturer specifications of the highest capacity cartridges available for color laser all-in-ones under $1,000 and mono laser all-in-ones under $500, as reported by Current Analysis, Inc. as of July 2006. Officejet Pro L7500/L7600/L7700 AiO CPP based on HP 88 Large Ink Cartridges (not included, sold separately) estimated street price and published yield. Results may vary. Estimated U.S. retail price. Actual price may vary. Simulated images.



Upgrading to Leopard?

Don't forget to feed the beast.

Upgrade your system memory with Mushkin PerfectMatch to get the most out of OS-X Leopard

It's a jungle out there when it comes to operating system upgrades and the ever increasing minimum system requirements for optimal performance. Now, with Mushkin PerfectMatch memory, you won't have to play guessing games and choose between the increased functionality of Leopard and your current version of OS-X.

Because every PerfectMatch Apple memory module is built from components carefully selected to provide true compatibility and match or exceed the performance of the original Apple memory installed at the factory. Which means you can tame the beast and confidently and easily upgrade the performance of your system to run Leopard.

mushkin **PerfectMatch**

Memory to match your system perfectly.

- ***Guaranteed, 100% Compatibility***
  Every PerfectMatch module is guaranteed to match or exceed the performance of the original memory installed at the factory.
- ***Save hundreds of dollars over the price of a new Mac***
  Mushkin PerfectMatch memory upgrades provide an easy and inexpensive way to optimize overall performance.
- ***Taking care of your company's road warriors?***
  Now you can cater to your laptop users' voracious appetite for RAM and enhance their mobile computing lifestyle.
- ***Your creative genius just met its match.***
  Upgrading with PerfectMatch memory increases performance in memory-hogging multimedia applications.

Now That Cat Can Run.

v i s i t u s o n l i n e a t w w w . m u s h k i n . c o m

North America

333 W Colfax Ave Ste 400
Denver, Colorado 80204
United States of America
tel: (800) 569-1868
fax: (303) 534-0827

Europe, Middle East & Africa

Raiffeisenstr.9
76872 Minfeld
Germany
tel: +49 7275-9888-13
fax: +49 7275-9888-69

Asia & Pacific

B-13-9, Megan Avenue II, No12
Jalan Yap Kwan Seng
50450 Kuala Lumpur
Malaysia
tel: +60-12-3157008

<m Get More.

MacMall

MacMall is the #1 Apple Direct Reseller &

Introducing MacBook

The world's thinnest & most elegant notebook!

MacBook™ Air

- Up to 1.8GHz Intel Core 2 Duo processor ■ Only 0.76" thin & weighs just 3lbs.!
- 80GB hard drive or 64GB solid-state drive ■ 802.11n Wi-Fi wireless
- 13.3" glossy widescreen LED backlit display ■ Full-sized QWERTY keyboard
- Built-in iSight™ video conferencing ■ Up to 5 hours of battery life

starting at $1744!* #7373085
Finance for $54/mo.

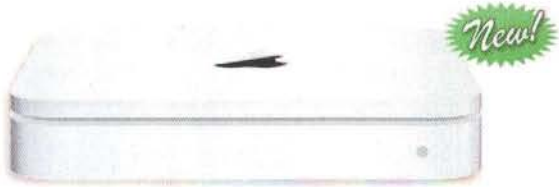
The amazing MacBook Air is thin enough to fit in a manila envelope! It's the ultimate blend of slipstream design and jet-powered productivity!

Only 0.76" thin!
Weighs just 3lbs.!
Up to 5 hours of battery life!

NEW Apple TV!

- Buy TV episodes and music videos from the iTunes Store and watch them on your standard or high-definition widescreen TV
- Rent movies online and watch them instantly
- Connects to your network at 802.11n wireless speeds or through an Ethernet connection
- Enjoy your computer's music collection, photo library and digital video on your home theater
- Access your .Mac Web Gallery or Flickr photo
- HDMI output ■ 1080p compatible

40GB only $224! #7185347



NEW Apple Time Capsule!

- Works with Time Machine in Mac OS X Leopard to back up everything on your system
- Automatic wireless or wired backup
- Functions as a full-featured AirPort® Extreme Base Station with 802.11n wireless
- Connect to multiple Macs and PCs
- Share a printer throughout your wireless network via the built-in USB 2.0 port
- 128-bit WEP encryption and NAAT firewall
- Available in 500GB and 1TB models

500GB only $294! #7373106

VMware Fusion for Mac
#7288411

~~$67.99~~ $47.99!
After $20 mfr. mail-in rebate. Price before rebate is $67.99. See below.†

Parallels Desktop 3.0 for Mac
#7257816

FREE!
After rebate with Apple computer purchase. See below.†

Apple Final Cut® Studio 2
#7236215

upgrade $448.99!

NEW! Microsoft Office 2008!

New versions of your favorite office applications with support for both Intel and PowerPC Macs!

Microsoft Office 2008 for Mac Home and Student Edition
only $129.98! #7352258

Allows for three installations from one box!

New!

*WE'LL BEAT ANY PRICE ON ANY APPLE BRAND PRODUCTS or competitive promotion on any Apple computer. We may require proof of dealer's pricing. Competitor must be an Apple Authorized Reseller. Product must be in stock at the competition. Advertised item must be identical to the one offered by MacMall and must be a new item in a factory sealed box. We reserve the right to limit quantity ordered. This offer does not apply to special, bonus or free offers, grand openings, special purchases or special buys, manufacturer's rebates, closeouts, clearances or to dealers' one-of-a-kind or other limited quantity offers. Nor does it apply to advertising errors made by any authorized dealer. Not open to dealers. Void where prohibited. Sorry, offer not valid in conjunction with other rebate offers. On multiple-item quotes: "We'll Beat Any Price" applies to the price of the entire order, not individual line items. Offer applies to pre-taxed prices only. Offer valid only at time of purchase, not after the fact. ■ UP TO $150 CASH BACK OFFER-Get up to $150 Cash Back via MacMall mail-in rebate with purchase of select computer models. Ends 3/15/08. ■ FREE PRINTER OFFER-Get an Epson Stylus C120 Color Inkjet Printer FREE after $70 and $10 mfr. mail-in rebates with purchase of any Apple computer. Price before

Authorized Reseller

the Mac pro's choice!

Air!

MacMall Exclusive Deals!

Get up to $150 cash back!*

FREE Parallels Desktop 3.0!*

FREE Epson Inkjet Printer!*

See below for details.

MacMall is your source for LCD and Plasma TVs!

NEW! Mac® Pro now with 8-core power!

- NEW! Up to 8-core performance with up to two Quad-Core Intel Xeon 5400 series processors at up to 3.2GHz
- NEW! ATI Radeon HD 2600 XT video graphics with 256MB GDDR3 memory
- NEW! PCI Express 2.0 graphics slot
- NEW! Four internal drive bays support up to 4TB of hard disk storage
- 16X SuperDrive™ with double-layer support (DVD±R DL/DVD±RW/CD-RW)
- Dual Gigabit Ethernet interfaces
- FireWire 800, FireWire 400 and USB 2.0

NEW Mac Pro starting at

$2173!* *Finance for $63/mo.*

intel Xeon

1-800-622-6255 macmall.com Source Code: MACTECH

Apple MacBook™ Pro

- Intel Core 2 Duo processing at up to 2.6GHz
- NVIDIA GeForce 8600M GT professional graphics
- 15" models with mercury-free LED-backlit display
- 17" model with optional 1920 x 1200 display
- Built-in iSight®, wireless and Gigabit Ethernet

~~$1994~~ – $150 mail-in rebate
starting at **$1844!*** #7254324

Apple iMac®

- Intel Core 2 Duo processor at up to 2.4GHz or Intel Core 2 Extreme processor at 2.8GHz
- Up to 500GB HD ■ Glossy widescreen display
- ATI Radeon HD graphics ■ Aluminum casing
- Built-in iSight®, wireless and Gigabit Ethernet

~~$1194~~ – $50 mail-in rebate
starting at **$1144!*** #7288171

Your iPod Superstore!

With every flavor of player and the best special offers your can find, MacMall is your source for everything iPod! And you can visit us online for the widest selection of accessories at the best prices!

Call for iPods in all colors & capacities!

#7297622 iPod® shuffle® (Blue) only $78.99
#7297625 iPod® nano (Silver) starting at $144
#7297631 iPod® classic (Silver) starting at $244

**Free case and shipping do not apply to iPod shuffle.

NEW iPod® touch!

32GB now available!

- Includes NEW software revision 1.1.3 with Mail, Maps, Stocks, Weather and Notes
- Wi-Fi Web browsing ■ Multi-touch interface
- View movies, photos and YouTube videos

Check our Web site for best price!

#7373111 iPod® touch 8GB $294
#7373112 iPod® touch 16GB $394
#7387505 NEW! iPod® touch 32GB $494

Adobe Creative Suite 3 Master Collection

Call for affordable upgrade paths!

#7228222

only **$2439.98!**

Apple Cinema Displays

#459836 20" Cinema Display® *was ~~$694~~ now* $594
#459837 23" Cinema HD Display *was ~~$994~~ now* $894
#459838 30" Cinema HD Display *was ~~$1994~~ now* $1794

Apple® AirPort® Extreme Base Station

802.11a/b/g/n

#7288377

only **$173.99!**

2x 5x

Mac OS® X Leopard™

#7355254

only **$109.98!**

#7355255 Family 5-Pack $179.99

rebates is $80. Ends 3/31/08. ■ FUSION OFFER-Price after $20 mfr. mail-in rebate with purchase of any Apple CPU or Mac OS X Leopard. Price before rebate is $67.99. Offer expires 6/30/08. ■ FREE PARALLELS DESKTOP 3.0 OFFER-Get Parallels Desktop 3.0 for Mac (#7257816) FREE after $20 mfr. and $60 MacMall mail-in rebates with purchase of any new Intel-based Apple CPU. Price before rebates is $80. Ends 3/31/08. ■ FREE CASE OFFER-After $5.99 MacMall mail-in rebate with purchase of any iPod touch, iPod classic or iPod nano. Price before rebate is $5.99. Ends 3/15/08. ■ FREE ENGRAVING OFFER-Get select touch, iPod classic, iPod nano and iPod shuffle models engraved for FREE when you purchase them through MacMall. No rebate required. There is a $9.99 charge for select iPods. Call or visit www.macmall.com/myipod to place your order. ■ ALL OFFERS VALID ONLY WHILE SUPPLIES LAST. ■ †For rebate terms and conditions, please visit our Web site and enter the applicable part number. Download rebate coupons at www.macmall.com/rebates.

Photofont

More than just one face of the web

CRAFTED BY FontLAB

WWW.FONTLAB.COM

In the old days, if you were designing web pages, you had two choices: **a)** make the type look good using a bitmap — and lose the ability to have your text indexed, searched, copied, or spell-checked, or **b)** use a generic web font.

That's history. With **Photofont** technology, your website can boast impressive typography, yet keep all the advantages of standard HTML text. Test it yourself at **http://www.photofont.com**

WEB TYPOGRAPHY

Photofont® WebReady™ lets you format web headlines and other short web texts with any fonts available on your computer without damaging the original html. So you can still search and copy the text and virtually every user will see your website the way you intend it. And search engines such as Google will like your pages, too!

PRINT TYPOGRAPHY

Our Photofont® plugins for **Adobe Photoshop®, Adobe InDesign® and QuarkXPress®** allow you to use photofonts in any print publication as well as on web pages.

COLOR TYPOGRAPHY

But **Photofont®** technology allows you to use more than just the normal fonts. You can also use photofonts with full color or grayscale, gradients, transparency and cool effects.

With **BitFonter™**, you can make photofonts out of any digital image. Perfect for illustrators, scrapbookers, photographers or calligraphers.

FONT EDITOR HEADQUARTERS

Fontlab Ltd. is the world leader in font editors and converters: **TypeTool™, TransType™, Fontographer™, FontLab Studio™, AsiaFon Studio™, BitFonter™, ScanFont™, and more.**

VISIT US AT **WWW.FONTLAB.COM** AND **WWW.PHOTOFONT.COM**

ARTICLES & DEPARTMENTS

Mac in the Shell

Data Manipulation with PHP

PHP provides easy access to MySQL on OS X

by Edward Marczak 8

Test-Driven Development Using AppleScript

Using testing frameworks to create more robust AppleScript applications

by Andy Sylvester 20

Modularity

Continuing our look at creating deployment images, with an intro to InstaDMG

by Philip Rinehart, Yale University 36

RubyCocoa

A new way to write Cocoa applications—Part 1

by Rich Warren 38

Grokking OS X's Undo Support

How to effectively use the Undo Manager built into Cocoa

by Marcus S. Zarra 60

The Road to Code

Nice and Gooey

Writing your first Cocoa GUI interface

by Dave Dribin 68

MacTech Spotlight

Eberhard Rensch

Founder, Pleasant Software 88

From the Editor

"Best practices." It's a phrase that's been on my mind lately. A lot. Possibly because I've had interactions with what I can only consider "worst practices." However, I really dislike the phrase altogether – there's no such thing, really. At least, not in the way it's typically presented. The best practices applicable to any given situation change with the situation, as no two are perfect matches. There are ways to try to make processes more efficient and predictable. Those are the tools you need in your toolbox. Reach for them often, and understand how they work. That way, you can turn any situation into one using "best practices."

Speaking of which, we have some articles this month that fall right into this category. **Philip Rinehart** leads us through an introduction to afp548's **InstaDMG**, a more formalized way of creating full OS installs through packages. Deployment of systems in a large environment is a huge challenge, and OS X has all of the software built-in that you'd need. You just have to decide how to use them. This month's **MacEnterprise** column tries to make your image-creation process a bit more consistent, modular and automated.

Following this, **Andy Sylvester** presents us with an article on **test-driven development using AppleScript**. On one level, it's a methodology – line up your tests in software before you write the code. While this article is specific to AppleScript, the method sits at a higher level than that and is interesting in its own right. If you're an AppleScripter, this article can show you how to build larger, more well-tested applications.

Leopard brought developers and sys admins a lot of new toys to work and play with. Xcode 3 alone brought great change. One of the beautiful updates is that more scripting systems have templates built-in that even interact with Cocoa. **Ruby** is one such language, and **Rich Warren** starts showing us how to take advantage of that. Check out the great sample application and get your Ruby groove on!

Dave Dribin, our resident Cocoa master, brings us further down the road and teaches you how to **write a GUI application**. If you've been following along, this is the first time that we're breaking out the GUI while learning Objective C, Xcode and Cocoa. Dave shows you the secrets to Interface Builder and hooking up your code to the GUI. It's the next stop on **The Road to Code**.

Also in the "best practices" category: mistakes. We all make them, and we need a way to recover. End-users have come to expect undo support in their favorite applications as a way to return to a state pre-mistake. But not every application supports undo. After reading **Marcus Zarra's "Grokking OS X's Undo Support,"** you'll wonder why. Marcus teaches the ins and outs of OS X's built-in Undo Manager, and how you can make it work in *your* application.

This month's **Mac In The Shell** continues to talk about PHP as a general scripting language, outside of any web use. This time, we specifically start to deal with **database support and data manipulation**. Moving data between systems, or even reporting on a data in a single database are common tasks. If you've been stymied by doing so in the past, read all about it here.

Last, but not least, the **MacTech Spotlight** shines on **Eberhard Rensch**, Founder of **Pleasant Software**. Before GarageBand, and before Podcast Producer, Pleasant Software brought us **Übercaster**, a podcast producing "studio." If you're deep into podcast, well, you're probably already using Übercaster. If you want to get into podcasting, Übercaster is a great way to go. Pleasant Software also makes some other cool tools, but, more importantly, we get some insight from the founder and main developer!

Until next month, keep working on *your* best practices.

Ed Marczak,
Executive Editor

Communicate With Us

Department E-Mails

Orders, Circulation, & Customer Service
cust_service@mactech.com

Press Releases
press_releases@mactech.com

Ad Sales
adsales@mactech.com

Editorial
editorial@mactech.com
(Authors only, no pr)

Accounting
accounting@mactech.com

Marketing
marketing@mactech.com

General
info@mactech.com

Web Site
http://www.mactech.com

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH.

We love to hear from you! Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

MACTECH®
The Journal of Macintosh Technology

A publication of XPLAIN CORPORATION

Canada Post: Publications Mail Agreement #41513541
Canada Returns to be sent to: Bleuchip International, P.O. Box 25542, London, ON N6C 6B2

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

by Edward Marczak

Data Manipulation with PHP

PHP provides easy access to MySQL on OS X

Introduction

Last month, we covered some of the basics of scripting with PHP. This month, we'll continue and talk about database access and string manipulation with PHP. Thanks to a pretty full install of PHP as the default in OS X, we don't have to worry about fetching, configuring, compiling and installing PHP – we can just get on with writing our script.

Working Environment

Last month, I mentioned that there are plenty of OS X-based editors that 'understand' PHP. This comes in the form of syntax coloring, appropriate code snippets, code completion and auto indentation. Before we continue, there are two more things that I should mention.

Coda, from Panic Software, is one of those editors – one that I should have mentioned last month. It's actually quite a bit more than a simple text editor: It's a full web development environment. I really mention it here because it has a way to access PHP documentation built right in (along with HTML, Javascript and more). Since PHP is such a popular web development language, as a web development tool, this made sense. At last check, though, access to this documentation was only available while you have an Internet connection.

I, on the other hand, am very much of an offline person. I want/need to accomplish things while on a plane, train, or wherever I may not have a good signal from some source, for whatever reason. Due to that, I've come to rely on the PHP Function Reference widget for Dashboard. Downloadable from here: http://code.google.com/p/phpfr/ (it has been "open sourced" since its introduction, and now lives as a Google Code project). Weighing in at close to 40MB, it's a little larger than your average widget, but that's the price you pay to carry around the entire PHP function set with descriptions. As a Dashboard widget, it lives out of sight until you need it. With a search function built-in, I tend to reach for this a lot. Highly recommended.

The Setup

I do a lot of data manipulation and migration work, and scripting languages are an essential part of the process. Perl, Ruby and Python are all good contenders for that position, but this particular column is about PHP! Our sample project this month will extract data from a MySQL database and give us a CSV file, ready for importing into another system. "But Ed," you say, "you can do *that* with MySQL alone!" While this is true, we aren't allowed to manipulate the data too drastically in that process.

Let's imagine that the data in our source system is stored with a single column for full name, but our target system wants separate first name and last name. Or that we need to go gather more data (like a zip+4 based on address) per record. There are hundreds of reasons you may need to filter the data as it goes from the database to a CSV file.

Connecting to the Database

One variable type that I left out of our discussion last month is *resource*. We looked at string, integer and other built-in types, but 'resource' really warrants its own discussion.

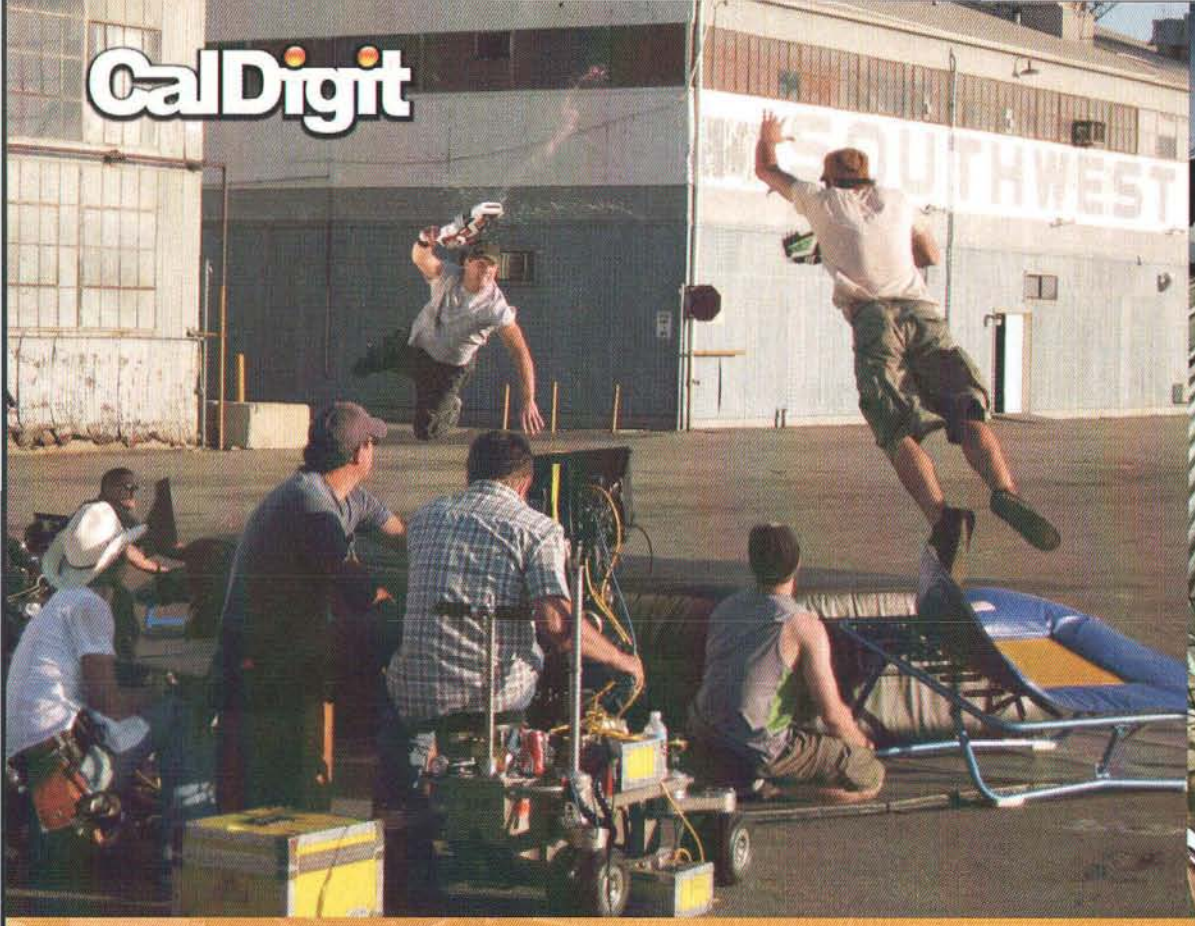
A variable of type "resource" holds a reference to some external resource. This may be a file on disk, a curl session to fetch a URL, a database connection or more. Again, it's just a reference, and you'll need to use functions that know how to access those resources via the resource variable that you pass to them.

First thing that our script needs to do is open a connection to the MySQL server. This happens with the `mysql_connect` command:

```
$connection =
mysql_connect("server.example.com","user","password");
```

The `mysql_connect` command passes back a resource if successful, and `FALSE` if there's an error. Let's get into good habits right away: check the value of `$connection` for `FALSE`. Thankfully, we have access to the error that the server passes back to us in the form of two functions: `mysql_errno()` and `mysql_error()`. These functions return the error code from the previous MySQL operation. We

CalDigit



"I can't afford hiccups when dealing with time sensitive deadlines! CalDigit's HDPro is reliable and delivers. It's the ideal solution for my editorial needs."

–Leo Zahn

HDPro

SCALABLE. SECURE. SCREAMING PERFORMANCE.

- 8 Drives: 2TB/2.5TB/4TB/6TB/8TB
- Supports RAID 0,1,5,6, and JBOD

LEO ZAHN - "THE UNCONVENTIONAL PATH"

Los Angeles based Commercial Director/Cameraman/Editor Leo Zahn has directed more than 500 commercials in the United States, Canada and Europe. A visual storyteller with a masterful eye for design, Leo Zahn has found challenges in many arenas: from automotive to food, from electronics to alcoholic beverages, from tabletop to over 200 toy commercials. His work has been rewarded with Clio and Cannes awards.

Most of his commercials require visual effects and multi-layered compositing, well before rough cuts are presented to the client. Having searched for a reliable and cost-effective RAID 5 solution, Leo decided on CalDigit's new HDPRO unit, which delivers screaming read/write rates of more than 380 MB/s. Ever since the HDPRO was installed in Leo's edit bay, it's been working non-stop. It's now the hub for all of his digital assets, scaled from uncompressed telecine footage to HD and 2K.

 http://www.caldigit.com

should follow our connect, or any database operation with something similar to this:

```
if (! $connection) {
    print "Connection to database failed with
".mysql_errno()." - ".mysql_error()."\n";
    die();
}
```

Simply, if `$connection` is not `TRUE` (zero, or, `FALSE`), we print out a message containing the MySQL error information, and then we halt operation of the script and exit with `die()`.

Also, looking at our initial connect string, what's wrong, or at least, what should send shivers up your spine? Yes, a password in a script. Unfortunately, that's just life in the big city, and you'll need to take precautions that allow only the right people to have access to the script. While you can put this information in external files and pull it in, or make your script get it from *somewhere*, the person that can read your file can also (typically) read how and where to get the password. Here be dragons, so, be conscious of this and plan for protection.

Once connected, we need to specify which database we're going to be working with. That's done with the `mysql_select_db()` function:

```
$select = mysql_select_db("customers");
…error checking here …
```

All subsequent MySQL functions operate on the selected database. If you're working with more than one server, you can optionally pass in the connection resource that you obtained from `mysql_connect()`:

```
$conn1=mysql_connect("server1","bob","s3kret");
$conn2=mysql_connect("server2","jane","h1dd3n");
$select1=mysql_select_db("sales",$conn1);
$select2=mysql_select_db("aggregates",$conn2);
```

…with the proper error checking, of course!

Once those preliminaries are complete, we can now query our database. While you can stuff a query into fewer lines, we're going to split it up in a more conventional manner. The `mysql_query()` function takes a string as an argument. Assigning the query to a variable makes for easier reading, easier variable substitution, and allows one to build dynamic queries. Let's start with a simple example:

```
$query="select uid,fname,lname from users";
$result=mysql_query($query);
```

`$query` is a simple string, and `$result` is of type resource. As with our earlier queries, if the execution fails, `mysql_query()` will return a Boolean `FALSE`, and our errors can be retrieved with `mysql_errno()` and `mysql_error()`.

`$result` is simply a resource, and points to the result set. We now have to use it to fetch the actual data, database row by database row. If you purposefully limited your query to a single

FAST, HIGH QUALITY MEMORY
OWC
FAST, HIGH QUALITY MEMORY

Add 4.0GB to your Apple® MacBook®, MacBook Pro, iMac®, or Mac® mini for $95.99

The Right Memory for ALL your Macs at Competitive prices from Other World Computing

www.owcmac.com/memory

Add up to 32GB to Apple Mac Pro from $40 per Gig!

OWC

Other World Computing™

BizRate 2007

Visit: www.owcmac.com or Call: 800.275.4576

row, you could now go fetch it with one line. However, more often than not, you're expecting multiple rows – sometimes thousands. A perfect job for the while loop that we introduced last month:

```
while ($row=mysql_fetch_assoc($result)) {
    print "Scanning ".$row['uid'].": ".$row['fname']."
".$row['lname']."\n";
}
```

That's a lot packed into a small space, so, let's break it down. The data fetching happens with this line:

```
$row=mysql_fetch_assoc($result)
```

You *could* manually call that over and over and get a new row from the database each call. Of course, for large result sets, a loop is the only feasible method. The `mysql_fetch_assoc()` function returns an *associative array*. Unlike a "plain" array, you can access values with a key that is a string. In our case, each column from the database becomes an entry in the array. We asked for the fields, "uid", "fname" and "lname", which makes `$row` become:

```
$row['uid']
$row['fname']
$row['lname']
```

..filled with the values from the current row retrieved from the database.

Within the body of the loop, there's a single line: a print statement. It's fairly ugly, though, isn't it? It happens to be pretty standard fare in PHP. The concatenation operator (".") 'glues' two strings together. So, now it should be pretty easy to break down, as we're printing some static text (the bits in quotes) and some dynamic text, filled in by the values in the `$row` array. Sample output would look like this:

```
Scanning 1052: Mike Bollinger
Scanning 1053: Laura Wilkinson
Scanning 1055: Joanne Moss
...
```

Now that we have access to our data, let's get it into a format usable by almost all other apps: CSV.

Data Manipulation

If you simply wanted a CSV file of a MySQL database table, there are many simple ways to do so. However, when moving data from one system to another, we often need to massage the original data into a new format. We get that opportunity in our `while` loop, where we're accessing each row of the table and before writing it out. PHP has many *string manipulation functions* to help us do anything our hearts (and brains) desire.

The first thing we may want to do is to simply re-assign a cleaned-up version of a string before writing it out. Consider this:

```
switch ($row['state']) {
    case "NY":
```

Apple Design Award
Best Mac OS X Leopard Application
2007 Runner-Up

You'll never go back to Quicken

iBank™

Your finances never looked so good.

Manage
- Bank Accounts
- Credit Cards
- Investments

Analyze
- Income
- Expenses

Plan
- Budgets
- Forecasts
- Schedule

Now available at these fine retailers:

MICRO CENTER The Ultimate Computer Store

Apple Store

TEKSERVE

amazon.com and you're done.

Office DEPOT

J&R JR.com 800 221 8180

OfficeMax

Fry's ELECTRONICS

IGG SOFTWARE

```
    case "new york":
    case "noo yrk":
        $state="New York";
        break;
    case "CA":
    case "calif.":
    case "cal":
        $state="California";
        break;
    default:
        $state="Unknown";
}
```

Rather than writing six separate `if` statements, I chose to use a single switch statement. This is a) cleaner code and b) a nice introduction to the `switch` statement, which I did not cover last month. Generally, the `switch` statement will test a single expression for the results you choose to test for. In the preceding example, we're examining the variable "`$row['state']`". Each `case` statement within the `switch` block acts as an "if" statement, and all code through the following `break` statement is executed. The code in the `default` block matches when no other `case` statements match. While the `switch` statement may contain virtually any expression, the match in each `case` statement can only be a simple type: string, integer or floating-point. You cannot evaluate arrays or objects here. For these more complex cases, you still must rely on an `if` statement.

The simple line doing all of the work here is this: `$state="..."`. We're just assigning a literal to our variable. We're even free to re-assign `$row['state']` if it suits our needs and style.

One of the coolest ways to tear up a string is the `explode()` function. As always, an example is best:

```
$animal="bear cat dog elephant";
$the_animals=explode(" ",$animal);
```

After this runs, $the_animals will be an array containing:

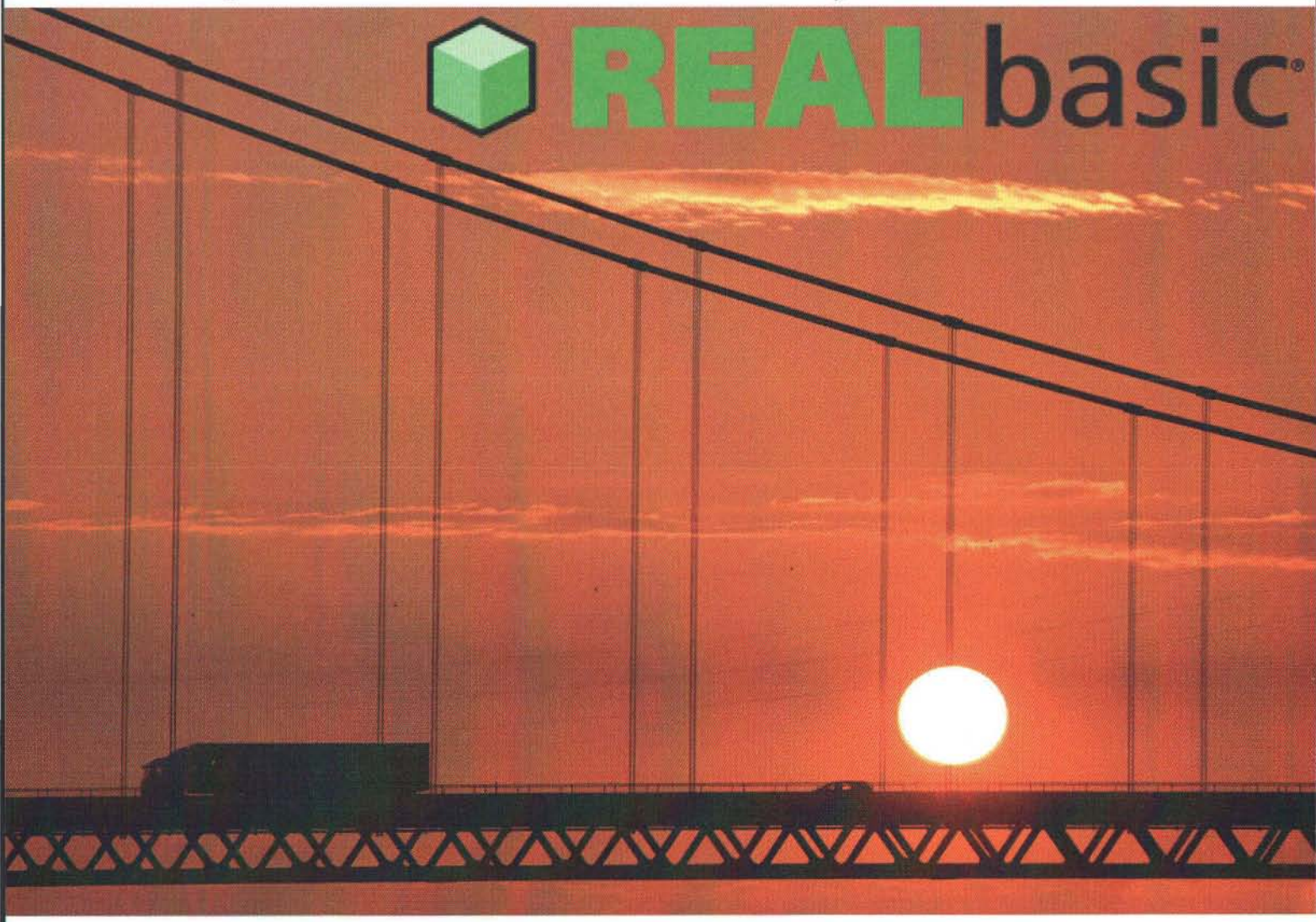
$the_animals[0]="bear"
$the_animals[1]="cat"
$the_animals[2]="dog"
$the_animals[3]="elephant"

explode() can split based on any *delimter*, not just a space. However, this is extremely useful for splitting up a full name. If you *know* that all entries in the database are in firstname-space-lastname format, then this will do what you want:

```
list($fname, $lname) = explode(" ",$fullname);
```

However, the reason you're probably involved is due to a more complex nature of the data. If *most* names are in two-name format, but some may contain a middle initial, too, then we have a few options. One would be to toss the middle initial, or include it in the first name. In that scenario we need to test how many elements were returned:

Cross-platform that really works.



Create your own software. Easily.

www.realbasic2008.com
Try REALbasic risk-free!
REALbasic includes a 90-day
money back guarantee.

REALbasic is the powerful, easy-to-use tool for creating your own software. At REAL Software, we call it a problem solver. You've probably said, "Wouldn't it be great if there was a little application that ..." REALbasic fills that blank.

REALbasic compiles native applications for Mac OS X, Windows and Linux from one set of source code. Each version of your software looks and works just as it should in each environment. You can even create a Universal Binary with one mouse-click.

```
$fullname="William S. Fatire";
$names=explode(" ",$fullname);
if (count($names) > 2) {
    $fname=$names[0]." ".$names[1];
    $lname=$names[2];
} else {
    $fname=$names[0];
    $lname=$names[1];
}

print "The names are:\n";
print "Firstname: $fname\n";
print "Lastname: $lname\n";
```

The `count()` function is introduced here and simply returns the number of elements in an array.

Naturally, our target database may have a field for middle initial, in which case, we can retain it and assign it appropriately.

Let's now imagine that we want to create a default user id for our new users. We can base this id on the user's real name. PHP includes some nice string slicing functions. If we want to base the user ID on the user's first initial plus last name, that's simple:

```
$uid=$fname[0].$lname;
```

PHP can access string elements by character when you supply the zero-based offset in square brackets. So, in this example, we're just grabbing the first character of `$fname`.

If you had a more complex uid-creation scheme, it'd be easy to handle, too. If you needed a uid that contained the first three letters from the family name, and the first two from the first name, PHP includes a nice sub-string function. Generically, it looks like this:

```
substr ($string, $start [,$length ])
```

This will return a string that starts at '$start' in $string, and runs until the end of the string supplied, or optionally, for $length. Back to our first-three, first-two uid scheme:

```
$uid=substr($lname,0,3).substr($fname,0,2);
```

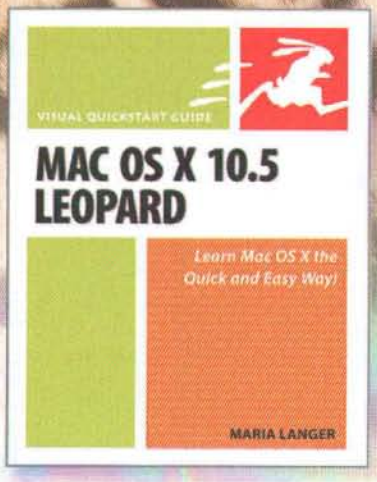
Nice, right? Of course, if you were really implementing this, you'd need to look for duplicates and also check for last names that may be shorter than 3 characters in total (like "Li").

All Together Now

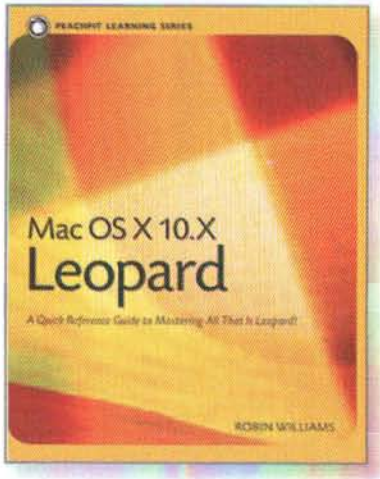
I'd like to put all of this together to create a code snippet that takes data from a database, gives space to manipulate the data, and output a CSV file. Some new elements will also be introduced here.

Listing 1: db2csv.php

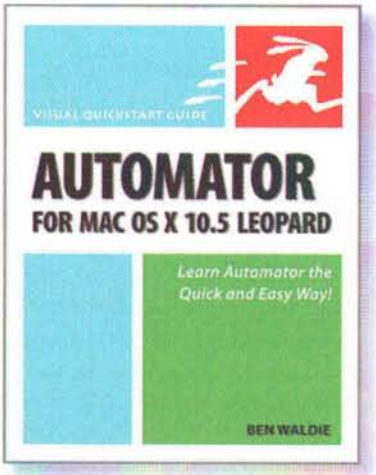
```
01 <?php
02
03 mysql_connect("127.0.0.1", "dbuser", "dbpass") or
04     die("Could not connect: " . mysql_error());
05 mysql_select_db("mydb");
06
07 $q="select * from user_profiles order by fid";
08 $r=mysql_query($q);
09 while ($row=mysql_fetch_assoc($r)) {
10  $pf[$row['fid']]=$row['title'];
11 }
12
```



Mac OS X 10.5 Leopard: Visual QuickStart Guide
ISBN: 0321496000
Maria Langer
Available now!

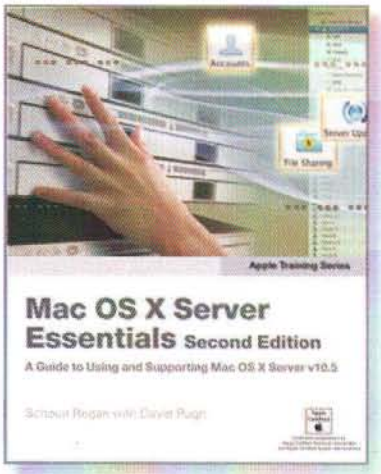


Mac OS X 10.5 Leopard: Peachpit Learning Series
ISBN: 0321502639
Robin Williams
Available now!



Automator for Mac OS X 10.5 Leopard: Visual QuickStart Guide
ISBN: 0321539354
Ben Waldie
Available now!

Apple Training Series: Mac OS X Support Essentials, Second Edition
ISBN: 0321489810
Available now!



Apple Training Series: Mac OS X Server Essentials, Second Edition
ISBN: 0321496604
Available now!

Are you ready for Leopard?

Learn the ins and outs of all the new features in Mac OS X 10.5 Leopard, including Time Machine, Spaces, QuickLook, Automator, and more. Whether you're an experienced Macintosh user ready to hone your trouble-shooting skills, or you're a beginner who needs help getting started, pick up a Peachpit book today to get the most out of the powerful features in Leopard.

BUY NOW AND SAVE!

Purchase or preorder your copy today and **save 35%** off the list price, plus free domestic shipping. Simply go to **www.peachpit.com/leopard** and enter coupon code **PP-LEOPARD** when you reach the checkout page.

Peachpit

Apple Certified

```
13 // Create the header
14 $curcsv="\"uid\",";
15 foreach ($pf as $key=>$val) {
16  $curcsv=$curcsv."\"".$val."\",";
17 }
18 $curcsv=substr($curcsv,0,strlen($curcsv)-1);
19 print "$curcsv\n";
20
21 $q="select * from users where status=1";
22 $r=mysql_query($q);
23 while ($row=mysql_fetch_assoc($r)) {
24 // Build csv for current user
25  $curcsv="\"".$row['uid']."\",";
26  foreach ($pf as $i=>$val) {
27      $q2="select value from user_profiles where fid=$i and
uid=".$row['uid'];
28      $r2=mysql_query($q2);
29      $row2=mysql_fetch_assoc($r2);
30      $curcsv=$curcsv."\"".$row2['value']."\",";
31  }
32  $curcsv=substr($curcsv,0,strlen($curcsv)-1);
33  print "$curcsv\n";
34 }
35
36 ?>
```

Newly introduced here is the `foreach` loop. `foreach` gives the programmer (*you*) an easy way to iterate over arrays. Generically, foreach looks like this:

```
foreach ($array as $value)
```

The loop interates once for each element of the array, and $value is updated accordingly. A variation to this (seen on line 26 in listing 1) also includes the current key – very useful for associative arrays where the key is text or other representation of the index. That variant is simply:

```
foreach ($array as $key=>$value)
```

Breaking down listing 1, lines 1-5 should look familiar: the opening PHP tag, and then try to connect to the database. Line 7 defines the first query to the first table, and line 8 executes that query against the selected database. The `while` loop starting at line 9 is pretty standard fare, but what's going on in the loop body?

On line 10, we're creating an associative array (`$pf`) using a variable as the key ("index"). In this case, we're using the result of the database fetch `$row['fid']`. Pretty slick.

Lines 14 through 17 take case of one small but significant part of writing out a CSV file: making the first line a header that describes the remaining columns. In our case, we can do that with the contents of the array we just created. Another `foreach` loop neatly solves that. In the body of this loop, we're creating a variable that will hold the current line of the CSV file. Each field is wrapped in quotes, and then followed by a comma.

Line 18 removes the trailing comma from `$curcsv`, and line 19 prints out $curcsv. Now for the bulk of work.

Lines 21 through 33 handle the main load of this program. Line 21 and 22 set up a new query and execute it. Line 23 brings back our now familiar while loop, fetching one database row at a time. Line 25 starts `$curcsv` anew each iteration, wrapping the proper value (`$row['uid']` in this case) in quotes followed by a comma.

Line 26 gets interesting: we use a new `foreach` loop to obtain more information about this particular uid for each of the values in $pf – by running a new query. We query and fetch again, and line 30 adds each field retrieved to `$curcsv` (wrapped and comma'd).

Finally, line 32 takes care of the trailing comma on `$curcsv` (because we blindly add it after each value), and line 33 prints the row.

Running this, or similar program, will simply dump all output to standard out – we're only using a `print` statement. You thought we were creating a CSV *file*? Well, we are! Everything in Unix is a file. Remember our shell redirectors? We can run this program as is and have the chance to visually inspect the output, and, when we're ready, simply redirect to a file:

```
php db2csv.php > users.csv
```

This is also a nice euphemism for, "I'm going to cover file manipulation next month!"

Conclusion

I'll continue to say: don't discount PHP as a general scripting language. It's fairly rapid development, has broad capabilities, and will typically be found across platforms (including versions for Windows – but you'll need to install it yourself). As mentioned, I'll cover some more aspects of scripting with PHP next month, including file manipulation.

Media of the month: Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture, by Jon Stokes. It's a very readable introduction to microprocessor arcitecture, and it's even current enough that it covers up through Intel's Core 2 Duo chips.

Until next time, enjoy your new scripting prowess!

References

Official PHP Documentation:
http://www.php.net/docs.php

About The Author

Ed Marczak is the Executive Editor for MacTech Magazine, and has been lucky enough to have ridden the computing and technology wave from early on. From teletype computing to MVS to Netware to modern OS X, his interest has always been piqued. He has also been fortunate enough to come into contact with some of the best minds in the business. Ed spends his non-compute time with his wife and two daughters.

Komodo IDE 4.3

Code. Debug. Edit. Share.
Let Komodo sweat the details.

Code Intuitively
Create great applications using dynamic languages and open technologies with Komodo IDE's award-winning multi-language editor. Code client-side and browser-side apps simply and naturally with advanced support for **Perl, PHP, Python, Ruby, Tcl, HTML, CSS, JavaScript, XML,** and more.

Work Smarter with Powerful Tools
Focus on what your code can do, not on fixing problems. Find errors quickly with Komodo IDE's superior remote and multi-threaded **debugging**, solve tricky regular expressions with the Rx Toolkit, and get instant feedback from syntax checking and syntax coloring—Komodo IDE is jam-packed with intelligent tools to speed development.

Build and Organize with a Team
Get your project together efficiently with **source code control** integration, integrated project manager, and multi-user support. Team development is more fluid with the security and flexibility of Subversion, CVS, and Perforce support. Share file templates, common configuration files, and toolboxes to unify and accelerate teamwork.

Develop Your Way
Automate, extend, customize: hack away!
With Firefox-style extensibility, highly configurable run commands, and powerful macros, you have complete control of your IDE. And you're covered wherever you want to work with Komodo IDE's flexible multi-platform licensing for **Windows**, **Mac OS X**, and **Linux**.

Download your free 21-day trial at: www.activestate.com/MacTrial

ActiveState

Test-Driven Development Using AppleScript

Using testing frameworks to create more robust AppleScript applications

by Andy Sylvester

Introduction

AppleScript can be used to automate many tasks on the Mac. However, compared to other scripting languages such as Perl, Python, and Ruby, it can seem somewhat simple and not as applicable for writing larger applications. These other languages also have testing frameworks that can be used for building and testing applications, and have good support for object-oriented programming. A testing framework for AppleScript called ASUnit has been developed, which provides a way to test AppleScript functions. Although AppleScript does not natively support OOP in the common way of Perl, Python, and Ruby, you can structure scripts to provide much of the same functionality. This article will introduce the concepts of test-driven development and demonstrate the use of the ASUnit testing framework.

Describing test-driven development

One of the significant changes in software development techniques in the past ten years has been a technique called "test-driven development". When using this development technique, the software developer writes a small amount of source code to test a function or feature in the application being developed. However, the software developer writes the test code before writing the actual application code. A summary of the technique is as follows:

Write some test code
Run the test and see that it fails
Write the source code that implements the feature for the test
Run the test again and see that it passes
Refactor or clean up source code

By writing test code before writing application code, the software developer creates a suite of tests that can be used at any time to check the functionality of the application. In addition, if changes are made to the application, the test suite can be run to see if the existing functions have been affected by the changes. To support test-driven development, many testing frameworks have been developed to assist software developers in creating, running, and managing tests. Kent Beck's work on testing frameworks for Smalltalk (http://www.xprogramming.com/testfram.htm) led to the development of the Junit testing framework for the Java language (http://www.junit.org). Since the appearance of Junit, testing frameworks have been developed for all programming and scripting languages, and even for web-based application development (see http://www.xprogramming.com/software.htm for a list of available frameworks). The second part of this article will introduce a testing framework for AppleScript called ASUnit. Before learning about this framework and how to use it, let us first look at some examples to see how writing tests can help with application development.

Exploring test-driven development

One way to add tests to an application is to use dialog boxes to give information on if a test passes or fails. A simple test to check a math operation could be written as follows:

```
display dialog "Test #1"
if 1 + 1 is equal to 2 then
    display dialog "Test 1 passes!"
else
    display dialog "Test 1 fails!"
end if
```

For this example, a set of dialog boxes would be presented, first announcing the title of the test, then the test

Premium Small Business Management and Accounting Software

...and now

Point of Sale for Mac
MYOB Checkout

MYOB

Mind Your Own Business. Smarter.

800 322 MYOB (6962)
www.myob-us.com

result. However, this could get awkward quickly, having to click on buttons to allow the program to run. Also, it is difficult to perform test-driven development, since the test code is intertwined with the application logic. Another approach is to create functions that contain application logic and then run the functions with test data. This would make it easier to test individual features without affecting other functions. Following the test-driven development philosophy, we will write a test function for application logic that checks for a specific input string. The following code demonstrates this technique:

```
script myNameGame
    CheckForMyName("Andy")
end script

run myNameGame
```

When this script is run in the Script Editor, a dialog box appears with the error message "«script myNameGame» doesn't understand the CheckForMyName message". This is to be expected, since the function `CheckForMyName` does not exist. So far, we are following the checklist of steps given above. Next, we add application logic for the `CheckForMyName` function:

```
on CheckForMyName(testGuess)
    if testGuess is equal to "Andy" then
        return "Your guess is correct!"
    else
        return "Nope! Try again!"
    end if
end CheckForMyName

script myNameGame
    CheckForMyName("Andy")
end script

run myNameGame
```

The script `myNameGame` calls the function `CheckForMyName` with an input string. The results from the test appear in the Results area of the Script Editor window. When this script is run, the expected result appears (`Your guess is correct!`). We have completed all but the last step of the our test-driven process (refactor and clean up). Since this logic is pretty simple, we will move on to adding some new features. Once again, we start with adding a test. The new feature to be added is that the function should check for the string "Andy" or "Bill". We can modify the myNameGame test to use "Bill" as the test string instead of "Andy". When we run the script, we get the failure result (Nope! Try again!). We can now add logic to check for both strings. Our function now looks like this:

```
on CheckForMyName(testGuess)
    if (testGuess is equal to "Andy") or ¬
        (testGuess is equal to "Andy")  then
        return "Your guess is correct!"
    else
        return "Nope! Try again!"
    end if
end CheckForMyName
```

Record. Edit. Play.

WireTap Studio's patent-pending LivePreview™ technology lets you hear your encoded audio before you record. Lossless master recording technology lets you play with compression settings, effects, and also edit your audio without any loss of quality or content.

It's a true breakthrough in audio recording technology.

WireTap Studio

Download your free trial now: http://www.AmbrosiaSW.com/MacTech

AMBROSIA®
SOFTWARE INC

winner macworld eddys

The Mac Observer EDITORS CHOICE AWARD 2008

Audio Units | Mac | Universal | Works with .mac | WireTap Studio *I'd Tap That!*

Macworld 1/2

WireTap Studio requires a Mac with an Intel, PowerPC G5, or PowerPC G4 (1GHz or faster) processor and Mac OS X 10.4 or later.

```
script myNameGame
    CheckForMyName("Bill")
end script

run myNameGame
```

When this script is run, the successful result appears again (`Your guess is correct!`). Now we can keep adding feature after feature using this test-driven development technique and make sure that the application logic is working correctly.

As more functions are developed, though, this simple test structure could be a burden to maintain. If each function to be tested had to be in its own file, this could result in many intermediate files for development. Also, the above structure is set up to only run one test at a time on a function. If running multiple tests were desired, the test script would have to be edited each time. Now that we have demonstrated the basics of test-driven development, let us look at a testing framework that can address the maintenance aspects of developing and running multiple tests on multiple functions.

using asunit

ASUnit is contained in a single AppleScript file. To install the program, download the latest version from the ASUnit website (http://nirs.freeshell.org/asunit/), unzip the file, and copy the file ASUnit.scpt to the Scripts folder in your Library folder or the Library/Scripts folder on the startup disk for your Mac. The remaining files are the README file for the application and a web page containing the ASUnit documentation (this page can also be found on the ASUnit website).

When creating an ASUnit test script, you must include a path to the location for ASUnit.scpt. Here is an example:

```
property parent : load script file¬
    (("Sylvester HD:Library:Scripts:") & "ASUnit.scpt")
```

Note that the path to the file needs to use the POSIX form with colons.

Next, we need to create a test fixture, or a framework, that will contain the tests that we wish to write. In ASUnit, this test fixture is an AppleScript which will contain other scripts within the main script. We can use the following structure:

```
script |accessing list|
    property parent : registerFixture(me)

    property empty : missing value
    property notEmpty : missing value
```

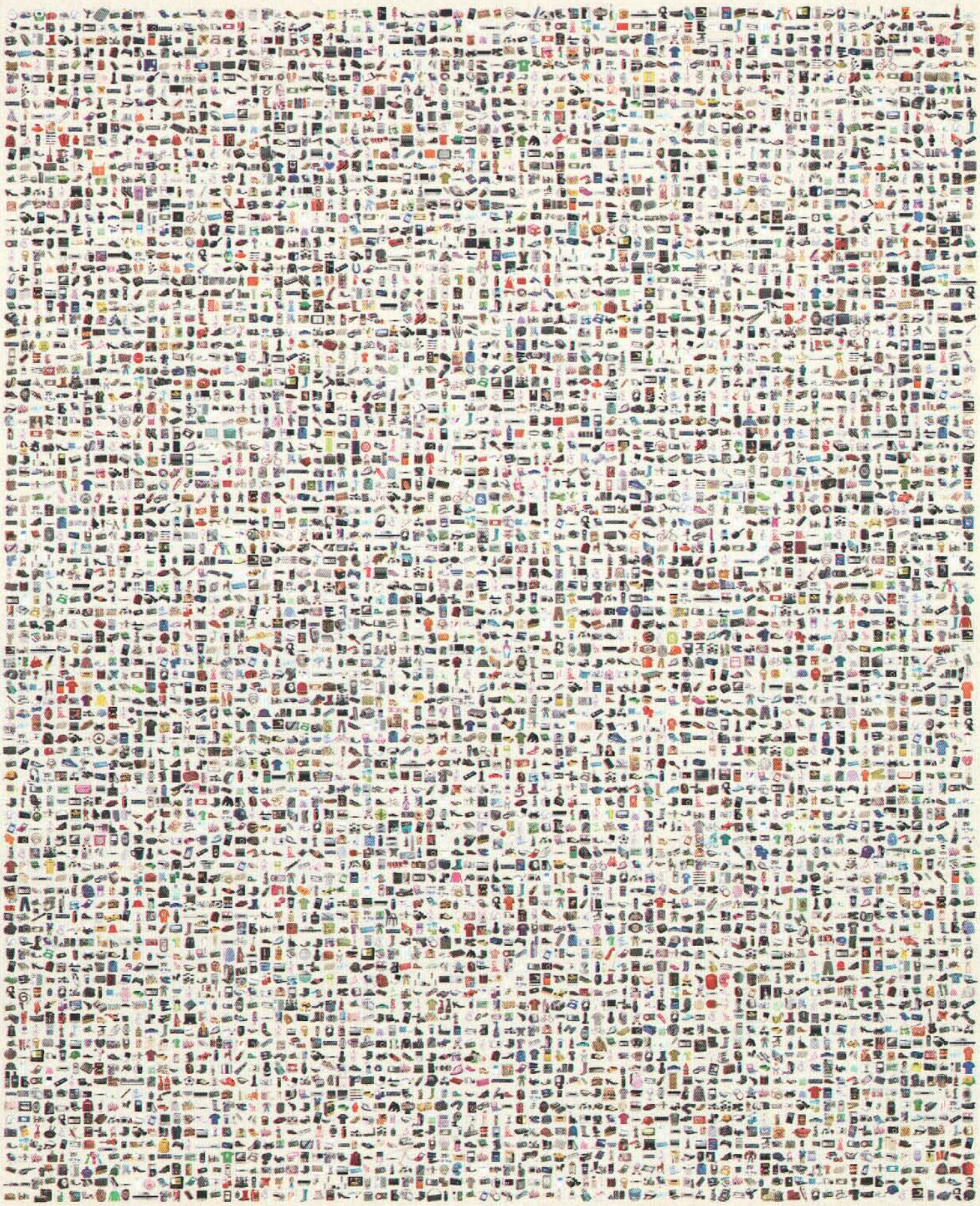
At the beginning of this script, the `registerFixture` script object is called. This script object is included in ASUnit.scpt. The accessing list script creates a property using the parent reference (which was included in the script file as the first line) to be able to access the `registerFixture`

THINGS MADE IN CHINA

COMPRESSED BY

STUFFIT DELUXE

SOFTWARE FOR MAC/PC



script object. Using the `me` keyword tells `registerFixture` that the `accessing list` script object is the fixture.

The last two lines define properties for objects that will be used in the test scripts that follow. For this example, two lists will be used (one that has no elements, and one that has at least one element).

When running ASUnit test scripts, there is a `setUp` script object which can be overridden with a local version to perform setup of objects for each test. This script will be called for each test script within the test fixture. For our example, we will create the two lists as follows:

```
on setUp()
    set empty to {}
    set notEmpty to {"foo", 1}
end
```

To perform a test, add another script which contains logic to perform some operation on the test objects. In this example, the objects we are manipulating are the lists `empty` and `notEmpty`.

```
script |add item|
    property parent : registerTestCase(me)
    set end of empty to "bar"
end
```

At the beginning of this script, the `registerTestCase` script object is called. This script object is also included in ASUnit.scpt. The `add item` script creates a property using the parent reference (which was included in the script file as the first line) to be able to access the `registerTestCase` script object. Using the `me` keyword tells `registerTestCase` that the add item script object is the test case to be registered. Next, the word "bar" is added to the end of the list `empty`, so that there is now an element within that list.

ASUnit provides a method, `should`, to check for a positive result for a condition. The method takes an AppleScript expression as the first argument, and an error message as the second argument. If the result of the expression is false, the error message will be displayed, otherwise the text "ok" will be displayed for that test. Add the following line to the add item script:

```
should(empty contains "bar", "no bar?!")
```

Since the previous line added the word "bar" to the list, this test should pass.

The final addition to this script will be logic to display the test results. ASUnit provides a script object called `makeTestSuite` to collect a number of test scripts into a single suite to be run. For this example, we only have one script. However, this function can be used with another ASUnit script object called `makeTextTestRunner` which will run all of the tests in a suite and create a window with the output of

Mac be nimble.
Mac be quick.

"We Make Mac Networking Child's Play."

Hello,

We're Small Tree Communications, The Mac networking experts.

Founded in 2003, by a group of talented networking and kernel engineers with high performance supercomputing experience, Small Tree Communications sells and supports high speed networks to make your work-flow run faster at a *surprisingly affordable* price.

- **SAVE TIME on really big file transfers with our across-the-board jumbo frame support**
- **SHARE all your stuff with all your friends off a platform you know and trust: An Apple Xserve**
- **SQUEEZE every bit of bandwidth out of your expensive RAIDS with Small Tree's Link Aggregation Compatible Switches**
- **BAN THE SAN: *Save your money, share your stuff***

To see all our cool toys, visit us at childsplay.small-tree.com

Receive a 10% DISCOUNT when you purchase any of the cool toys on our website:
Just enter the promo code *MacTech* at checkout.

Small Tree COMMUNICATIONS

Small Tree Communications 7300 Hudson Blvd., Suite 110, Oakdale, MN 55128 866.STC.4MAC email: childsplay@small-tree.com

the tests. After adding logic to call these functions, our first ASUnit test script will be as follows in Listing 1:

```
property parent : load script file ¬
    (("Sylvester HD:Library:Scripts:") & "ASUnit.scpt")
property suite : makeTestSuite("My Tests")

script |accessing list|
    property parent : registerFixture(me)

    property empty : missing value
    property notEmpty : missing value

    on setUp()
        set empty to {}
        set notEmpty to {"foo", 1}
    end setUp

    script |add item|
        property parent : registerTestCase(me)
        set end of empty to "bar"
        should(empty contains "bar", "no bar?!")
    end script
end script
run makeTextTestRunner(suite)
```

After typing the above AppleScript code in a Script Editor window, click the Run button to execute the tests. You should see a new window open and display the following text:

```
My Tests

accessing list - add item ... ok

Ran 1 tests in 0 seconds.  passed: 1  skips: 0  errors: 0
failures: 0

OK
```

The one test (`add item`) passed - hooray! But what if a different word than "bar" had been added to the list empty? If we change "bar" to "bat" in the line adding the text to the list, we get the following test results:

```
My Tests

accessing list - add item ... FAIL

FAILURES
-----------------------------------------------------------
test: accessing list - add item
message: no bar?!
-----------------------------------------------------------

Ran 1 tests in 0 seconds.  passed: 0  skips: 0  errors: 0
failures: 1

FAILED
```

Now you know what a test failure looks like. Once you have looked at the test results, you can click on the red Close button to close the window, then click on the "Don't Save" button to complete closing the window.

Now that a test fixture has been created, you can add more tests. Also, within a test, you can have multiple "`should`" statements to perform more than one check. Listing 2 shows an addition to the previous script:

```
property parent : load script file ¬
    (("Sylvester HD:Library:Scripts:") & "ASUnit.scpt")
property suite : makeTestSuite("My Tests")

script |accessing list|
    property parent : registerFixture(me)

    property empty : missing value
    property notEmpty : missing value

    on setUp()
        set empty to {}
        set notEmpty to {"foo", 1}
    end setUp

    script |add item|
        property parent : registerTestCase(me)
        set end of empty to "bar"
        should(empty contains "bar", "no bar?!")
    end script

    script |add same item|
        property parent : registerTestCase(me)
        set end of notEmpty to "foo"
        should(last item of notEmpty is "foo", "first foo
vanished?!")
        should(first item of notEmpty is "foo", "where is
last foo?!")
    end script
end script
run makeTextTestRunner(suite)
```

Running this script give the following results:

```
My Tests

accessing list - add item ... ok
accessing list - add same item ... ok

Ran 2 tests in 1 seconds.  passed: 2  skips: 0  errors: 0
failures: 0

OK
```

Since both of the `should` statements were satisfied, the single "ok" message was printed. If either or both conditions had failed, their respective error messages would have been printed.

Testing an AppleScript class

Now that we have a working test script, we can start experimenting with adding some functions to test. In AppleScript, we can organize functions and data within a script to be able to create classes and objects like other languages. As an example, let's look at creating a class for storing calendar dates. Add the following script to Listing 2

```
script CalendarDate
    -- CalendarDate has three properties - day, month, and
year.
    property calendarDay : 0.0
```

Network, Server and Appliance Monitoring.

For Mac OS X

Xserve RAID

Xserve (Intel & G5)

Airport

Lithium Network Monitoring Platform

Lithium can now monitor your Xserve, Xserve RAID, Qlogic switches, Airports, Mac OS X Server... and everything else in your network.

www.lithiumcorp.com

lithium Corp.

```
    property calendarMonth : 0.0
    property calendarYear : 0.0
    - Sets the calendarDay property to the value passed to
it.
    on SetDay(theDay)
        set calendarDay to theDay
    end SetDay
    - Sets the calendarMonth property to the value passed to
it.
    on SetMonth(theMonth)
        set calendarMonth to theMonth
    end SetMonth
    - Sets the calendarYear property to the value passed to
it.
    on SetYear(theYear)
        set calendarYear to theYear
    end SetYear
    - Returns the value of the calendarDay property.
    on GetDay()
        return calendarDay
    end GetDay
    - Returns the value of the calendarMonth property.
    on GetMonth()
        return calendarMonth
    end GetMonth
    - Returns the value of the calendarYear property.
    on GetYear()
        return calendarYear
    end GetYear
    on InitializeDate(myDay, myMonth, myYear)
        SetDay(myDay)
        SetMonth(myMonth)
        SetYear(myYear)
    end InitializeDate
end script
```

We can imitate a class by creating a top-level function, declaring properties for object data, and creating functions to set the properties and get their values. To create script objects derived from this class description, we can use the copy command to make copies of the scripts, which will make copies of all of the functions and properties of CalendarDate. We can write the **setUp** function as:

```
on setUp()
    copy CalendarDate to firstDate
    copy CalendarDate to secondDate
    - Set the values for the first date.
    tell firstDate to InitializeDate(21, 9, 2007)
    - Set the values for the second date.
    tell secondDate to InitializeDate(25, 9, 2005)
end setUp
```

Now that we have two objects (**firstDate** and **secondDate**) and have initialized them, we can use the functions of the **CalendarDate** class to check the values of the two objects.

```
script |CheckSameMonth|
    property parent : registerTestCase(me)
    set p to firstDate's GetMonth()
    set d to secondDate's GetMonth()
    should(p is equal to d, "month not equal!")
end script

script |CheckDifferentMonth|
    property parent : registerTestCase(me)
```

```
        set p to firstDate's GetMonth()
        set d to secondDate's GetMonth()
        should(p is not equal to d, "month is the same!")
    end script
```

Our finished script now looks like this (Listing 3):

```
property parent : load script file ¬
    (("Sylvester HD:Library:Scripts:") & "ASUnit.scpt")
property suite : makeTestSuite("My Date Tests")

script |DateTests|
    property parent : registerFixture(me)

    property firstDate : missing value
    property secondDate : missing value

    script CalendarDate
        -- CalendarDate has three properties - day, month,
and year.
        property calendarDay : 0.0
        property calendarMonth : 0.0
        property calendarYear : 0.0
        -- Sets the calendarDay property to the value passed
to it.
        on SetDay(theDay)
            set calendarDay to theDay
        end SetDay
        -- Sets the calendarMonth property to the value
passed to it.
        on SetMonth(theMonth)
            set calendarMonth to theMonth
        end SetMonth
        -- Sets the calendarYear property to the value passed
to it.
        on SetYear(theYear)
            set calendarYear to theYear
        end SetYear
        -- Returns the value of the calendarDay property.
        on GetDay()
            return calendarDay
        end GetDay
        -- Returns the value of the calendarMonth property.
        on GetMonth()
            return calendarMonth
        end GetMonth
        -- Returns the value of the calendarYear property.
        on GetYear()
            return calendarYear
        end GetYear
        on InitializeDate(myDay, myMonth, myYear)
            SetDay(myDay)
            SetMonth(myMonth)
            SetYear(myYear)
        end InitializeDate
    end script

    on setUp()
        copy CalendarDate to firstDate
        copy CalendarDate to secondDate
        -- Set the values for the first date.
        tell firstDate to InitializeDate(21, 9, 2007)
        -- Set the values for the second date.
        tell secondDate to InitializeDate(25, 9, 2005)
    end setUp

    script |CheckSameMonth|
        property parent : registerTestCase(me)
        set p to firstDate's GetMonth()
        set d to secondDate's GetMonth()
        should(p is equal to d, "month not equal!")
    end script
```

```
    script |CheckDifferentMonth|
        property parent : registerTestCase(me)
        set p to firstDate's GetMonth()
        set d to secondDate's GetMonth()
        should(p is not equal to d, "month is the same!")
    end script
end script
run makeTextTestRunner(suite)
```

After adding these tests, we get the following results:

```
My Date Tests

DateTests - CheckSameMonth ... ok
DateTests - CheckDifferentMonth ... FAIL

FAILURES

test: DateTests - CheckDifferentMonth
message: month is the same!

Ran 2 tests in 0 seconds.  passed: 1  skips: 0  errors: 0
failures: 1

FAILED
```

Since both **firstDate** and **secondDate** had the same value for the **calendarMonth** property, the **CheckSameMonth** test passed while the **CheckDifferentMonth** failed (since both month values were the same). We can add additional tests to check the day and year values as follows:

```
    script |CheckSameDay|
        property parent : registerTestCase(me)
        set p to firstDate's GetDay()
        set d to secondDate's GetDay()
        should(p is equal to d, "day not equal!")
    end script

    script |CheckDifferentDay|
        property parent : registerTestCase(me)
        set p to firstDate's GetDay()
        set d to secondDate's GetDay()
        should(p is not equal to d, "day is the same!")
    end script

    script |CheckSameYear|
        property parent : registerTestCase(me)
        set p to firstDate's GetYear()
        set d to secondDate's GetYear()
        should(p is equal to d, "year not equal!")
    end script

    script |CheckDifferentYear|
        property parent : registerTestCase(me)
        set p to firstDate's GetYear()
        set d to secondDate's GetYear()
        should(p is not equal to d, "year is the same!")
    end script
```

Running all of the tests together gives the following results:

```
My Date Tests

DateTests - CheckSameMonth ... ok
DateTests - CheckDifferentMonth ... FAIL
DateTests - CheckSameDay ... FAIL
DateTests - CheckDifferentDay ... ok
DateTests - CheckSameYear ... FAIL
DateTests - CheckDifferentYear ... ok

FAILURES

test: DateTests - CheckDifferentMonth
message: month is the same!

test: DateTests - CheckSameDay
message: day not equal!

test: DateTests - CheckSameYear
message: year not equal!

Ran 6 tests in 2 seconds.  passed: 3  skips: 0  errors: 0
failures: 3

FAILED
```

We can see that the test results show that the month is the same for the two objects **firstDate** and **secondDate**, but that the day and year are not the same.

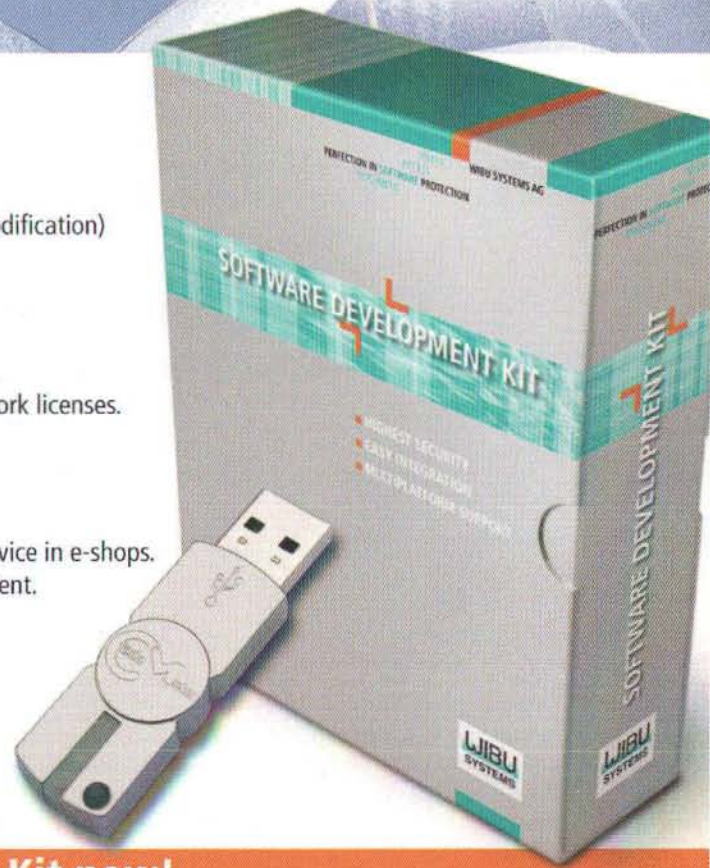
To conclude this example using ASUnit, we will show how to separate the program code from the test code. Copy the script **CalendarDate** to another file and call it Date.scpt. Next, save the above script as DateTest.scpt, delete the **CalendarDate** script and add another property at the top of the script to load the program code from another file. Finally, we need to modify references to **CalendarDate** in the test script to include the property added at the top of the file. After making these changes, the test script looks like this:

```
property parent : load script file ¬
    (("Sylvester HD:Library:Scripts:") & "ASUnit.scpt")
property lib : load script file ¬
    (("Sylvester HD:Test:") & "Date.scpt")
property suite : makeTestSuite("My Date Tests")

script |DateTests|
    property parent : registerFixture(me)

    property firstDate : missing value
    property secondDate : missing value

    on setUp()
        copy lib's CalendarDate to firstDate
        copy lib's CalendarDate to secondDate
        -- Set the values for the first date.
        tell firstDate to InitializeDate(21, 9, 2007)
        -- Set the values for the second date.
        tell secondDate to InitializeDate(25, 9, 2005)
    end setUp

    script |CheckSameMonth|
        property parent : registerTestCase(me)
        set p to firstDate's GetMonth()
        set d to secondDate's GetMonth()
        should(p is equal to d, "month not equal!")
    end script

    script |CheckDifferentMonth|
        property parent : registerTestCase(me)
```

CodeMeter®

No.1

in Software and Document Protection!



Highest Security

- Vendor selectable secret and private key.
- Strong encryption algorithms with AES 128-bit and ECC 224-bit.
- Best-in-class tools for automatic protection (envelope, without source modification) for Win32, Win64, .NET, Java and MacOS X Universal (PPC, Intel).

Best Flexibility

- More than 1000 independent licenses can be protected by one CM-Stick.
- One versatile hardware key for all license models including floating network licenses.
- Multi platform support including Windows, MacOS X and Linux.

New Distribution Channels

- License transfer by SOAP based CM-Talk or file based Field Activation Service in e-shops.
- Multiple-purpose, including protecting low cost software and digital content.

Unique End User Advantages

- First and smallest dongle with up to 2 Gbyte flash drive.
- No drivers necessary – can be used without administrator rights.
- CM Password Manager, secure virtual drive and secure login.

Order your Free Software Development Kit now!
Phone 1-800-6-GO-WIBU | order@wibu.us

2007 SIIA //CODiE// FINALIST
2005 SIIA //CODiE// FINALIST
2006 SIIA //CODiE// FINALIST

Mac

Universal

WIBU-SYSTEMS submitted the CodeMeter Password Manager and the CodeMeter SDK for the **Apple Design Awards 2007.**

WIBU SYSTEMS

WIBU®-SYSTEMS USA Inc.
110 W Dayton Street
Edmonds, WA 98020, USA
www.wibu.com
info@wibu.us
Tel: 1.800.6.GO.WIBU
1.425.775.6900
Fax: 1.206.237.2644

```
        set p to firstDate's GetMonth()
        set d to secondDate's GetMonth()
        should(p is not equal to d, "month is the same!")
    end script

    script |CheckSameDay|
        property parent : registerTestCase(me)
        set p to firstDate's GetDay()
        set d to secondDate's GetDay()
        should(p is equal to d, "day not equal!")
    end script

    script |CheckDifferentDay|
        property parent : registerTestCase(me)
        set p to firstDate's GetDay()
        set d to secondDate's GetDay()
        should(p is not equal to d, "day is the same!")
    end script

    script |CheckSameYear|
        property parent : registerTestCase(me)
        set p to firstDate's GetYear()
        set d to secondDate's GetYear()
        should(p is equal to d, "year not equal!")
    end script

    script |CheckDifferentYear|
        property parent : registerTestCase(me)
        set p to firstDate's GetYear()
        set d to secondDate's GetYear()
        should(p is not equal to d, "year is the same!")
    end script
end script
run makeTextTestRunner(suite)
```

When we run the test script DateTest.scpt from the Script Editor, we get the same test results as the combined file in the last section.

Conclusion

Using the concepts of test-driven development, you can build and test your application as you go to make sure that it works the way you want. Using the ASUnit testing framework, you can create a suite of tests to serve as a check of your application logic whenever you make updates. The tests you create give you the freedom to refactor your code while still being able to ensure that your application works as expected. In the second part of this series, we will develop a complete application using test-driven development and the ASUnit testing framework.

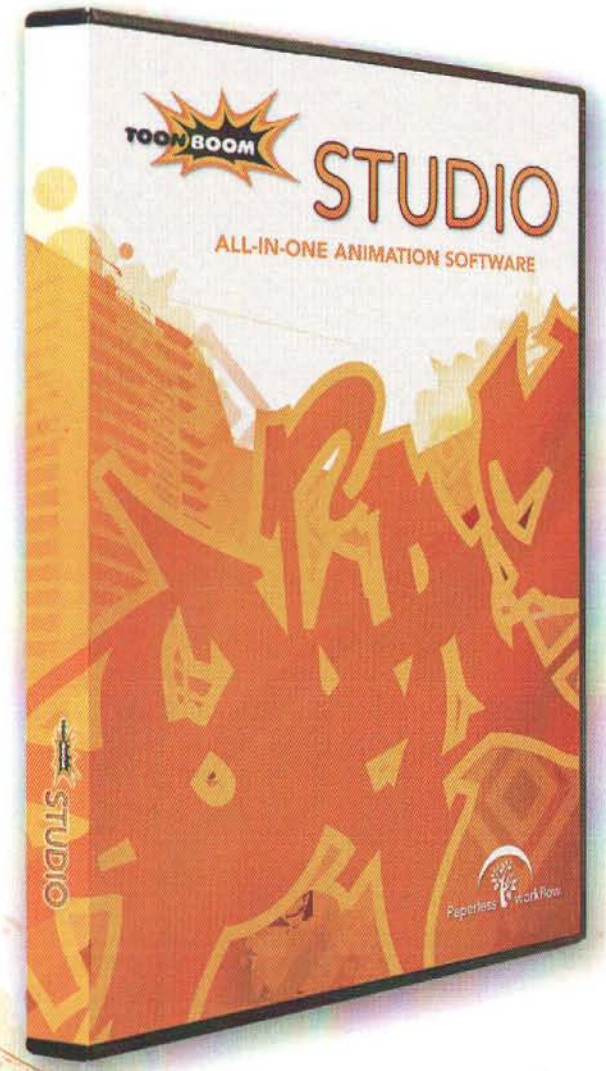
About The Author

Andy Sylvester is an aerospace engineer who has worked in software development for over twenty years. His interests include web applications, the use of computers in music composition, and software development techniques. He has a weblog at www.andysylvester.com ***where he writes on these subjects. You can reach him at*** andy@andysylvester.com.

ANIMATE WITH EASE!

TOON BOOM STUDIO

ALL-IN-ONE ANIMATION SOFTWARE



The only start to finish animation software toolset designed for creative and professional users.

- Draw, scan or import artwork into system
- Paint artwork automatically
- Lip-sync mouth movements to voice tracks
- Animate in conjunction with sound, effects
- Access libraries to use and reuse content
- Create motion and camera moves in 3D
- Apply special effects
- View animation in real time
- Publish to TV, HDTV, video, web (SWF, QT)

Teacher's guide and animation lessons available

toonboom.com

Modularity

Continuing our look at creating deployment images, with an intro to InstaDMG

By Philip Rinehart, Yale University

MacEnterprise.org
Mac OS X enterprise deployment project

MODULARITY

Recently, the Macenterprise list has been talking quite heavily about a topic from last month's article: image modularity. This month, let's take a look at new ways of creating images, focusing on a new tool from afp548.com, InstaDMG. The idea here is that any image can be created quickly, is not machine dependent, and can be deployed across the entire population of managed machines.

INSTADMG

InstaDMG, from afp548.com, starts to address the issues that we talked about in previous articles: scalability, modularity, and flexibility. At its core, a large shell script drives this solution. By using a shell script to drive the process, it is quite transparent, and can be easily understood. The first step to begin using the process is to insert a retail version of Leopard and creating a disk image of it. Wait, doesn't the break the cardinal rule of always using the disks that came with the machine. It does, but read on, and you'll begin to understand why you can break this rule with InstaDMG. At the time of this writing, once the dmg has been created, drop it into the **BaseOS** folder. You can keep the name as **Mac OS X Install DVD**.

Next, download **each** installation package from the Apple download site. As you are working off the **retail** version of Leopard, each and every download has to be applied to the image. This is where the beauty of InstaDMG begins to come through. Take all of the updates, and place them one at a time in individual folders in the AppleUpdates folder of the InstaDMG tree. There is one caveat, order is important for some packages, and you will have to know that when you create the tree. How do you know? It's a bit of a black art, in that there is nothing that will definitively give you that answer. In general, I've resorted to taking a base installation, taking a base install, and applying all software updates from the command line, paying attention to order of application.

After all of the Apple updates have been ordered, and placed in their correct folders, do the same for any software that you want to add to the base image. Again, placing the installation packages in folders controls order. Note though, that **everything** has to be packaged, as InstaDMG relies on being driven from the command line installer. These means no VISE installers, no ZeroG installers, nothing but Apple packages. Fortunately on Leopard, this task isn't as difficult as it once was thanks to the new Packagemaker snapshot option. Some packages **may also** need to be repackaged, as they aren't installed correctly from the command line. I ran into this straight away, as the CiscoVPN installer is a package, but doesn't install correctly from the command line. After repackaging, it was much happier and did work.

Automagic

One of the interesting things that recently was added is a new feature for the installer in Leopard. It allows for item selection directly with the use of an XML file. How does it work? In simple terms, an xml file is fed to `installer` at the command line, and options are selected or deselected. I'll summarize an excellent post from Patrick Fergus found on the afp548 forums, http://www.afp548.com/forum/viewtopic.php?showtopic=18907.

First, run the installer from the command line in the following way:

```
sudo installer -showChoicesXML -pkg my.pkg
```

This dumps an XML list showing all of the choices available in any package installer. Run this command on the OSInstall.pkg from an OS X Installation. Look for the xml tag, choiceIdentifier. Note how they are formed, they should look somewhat similar to the choices provided in a GUI installation dialog. The choice should show as being selected using 1 for selected or 0 for unselected.

Now, create a plain text file, with an array of strings, like so:

```
<array>
<string>choice1</string>
<string>choice2</string>
</array>
```

Save it as an xml file. One quick note, you can select/deselect options, even nested ones. Enter the option as many times as you need to click the option to enable it in the GUI. It sounds a bit more complicated than it is. After you have the xml file, check your options using the following command:

```
sudo installer -showChoicesAfterApplyingChangesXML my.xml -
pkg my.pkg
```

Compare it to the first command above. If the installer doesn't reflect the changes that you entered, then your xml file is incorrect. How does this new ability apply to InstaDMG? Creating an InstallerChoices.xml file can control the options that are installed with the operating system during image creation. Place this file at the root level of the directory. Here's how mine looks:

```
<array>
    <string>AdditionalFonts</string>
    <string>X11</string>
</array>
```

This file installs the Additional Fonts package, as well as the X11 package. Right now, InstaDMG only deals with an InstallerChoices.xml file for the Base OS. In the future, it will accommodate xml files for any other package that can use options.

Last steps

Once all of the packages and updates have been added, it really is simply a matter of running the bash shell script in the InstaDMG package. Launch it, and it will create an ASR scanned image in a rather short period of time. Note, the speed at which the image is created is entirely dependent on the computer that is creating the image. After the ASR image has been created, use whatever method you have to apply the image to a machine. In my usage of this product, I was able to build an image that I could apply to a MacBook Air! Pretty darn cool!

One last note this month, even though I've talked about InstaDMG, there are other programs that are attempting to do similar things. You could also use System Image Utility or PKGImage to do similar things. I just find InstaDMG to be straightforward and simple. With it, I no longer have to maintain a "build" sheet, and any of my staff can use the process – a major plus. Until next month, see you on the lists!

About The Author

Philip Rinehart is co-chair of the steering committee leading the Mac OS X Enterprise Project (macenterprise.org) and is the Lead Mac Analyst at Yale University. He has been using Macintosh Computers since the days of the Macintosh SE, and Mac OS X since its Developer Preview Release. Before coming to Yale, he worked as a Unix system administrator for a dot-com company. He can be reached at: philip.rinehart@yale.edu.

The MacEnterprise project is a community of IT professionals sharing information and solutions to support Macs in an enterprise. We collaborate on the deployment, management, and integration of Mac OS X client and server computers into multi-platform computing environments.

RubyCocoa

A new way to write Cocoa applications-Part 1

by Rich Warren

I admit it. I have a soft spot for Ruby. That probably comes as no surprise, if you've read my earlier articles (particularly *Introduction to Ruby on Rails* and *Ajax on Rails*, both available online at www.mactech.com). Needless to say, I'm quite giddy with Leopard's scripting language support. Leopard has elevated Python and Ruby to. . .um. . .not first class citizens. Not quite. But they make a strong second-class showing.

In fact, my biggest complaint comes from the terminology. Apple's own documentation refers to both Ruby and Python as scripting languages. Scripting Languages? Sure, they are both interpreted languages, but the word "scripting" makes them sound like limited, little things. Trust me, you can use these languages to do a lot more than just write scripts. We have two, full-blown, dynamic, object oriented programming languages, and Leopard puts their power at our fingertips.

New Ruby and Python Features

Ruby and Python are not new to OS X. Tiger shipped with both languages installed (though, if you've read my previous articles, you know that the Tiger version of Ruby kinda sucked). Leopard, however, kicks the support up a notch. They've invested a lot of time into getting the details right. While they may not always succeed, I appreciate the effort.

For example, Xcode comes with templates for a variety of Ruby and Python projects. Syntax highlighting and code completion work as expected. Most importantly, Leopard integrates both languages more tightly into the operating system. Both include a bridge to the Objective-C runtime, and both can communicate with scriptable applications.

The Bridge to Objective-C

Leopard ships with the popular RubyCocoa and PyObjC libraries already installed. Developers can use these libraries to write Cocoa applications in either Ruby or Python, respectively. Both languages have access to Leopard's core technologies, including Core Data, Bindings and Document-based applications. These libraries even support the new rock-star frameworks like Core Animation.

But, why would you want to use Ruby or Python? Some might say they're addictive; once you start using them it's hard to go back (trust me, I use Java for my day job). But, you can find other reasons as well. Both Ruby and Python are very expressive languages. You can get a lot of work done with very little code. This makes them ideal choices for rapid development and prototyping.

Additionally, Objective-C, Ruby and Python share many common concepts and design choices. They are all dynamic, object-oriented languages. Ruby and Objective-C in particular, were both heavily influenced by Smalltalk. This common ground helps us coordinate our code across the different languages.

And we can freely mix our code. We can use Ruby subclasses of Objective-C classes, or Python delegates for Objective-C objects. We can transparently call one language from the other. This gives us more power and more flexibility than any one language would have on its own. We have access to each language's libraries. We can exploit their individual strengths, using one language to spackle over the other's weaknesses.

Unfortunately, Cocoa seems to have a one-bridge-at-a-time rule. Mixing either Ruby or Python with Objective-C works just fine. But mixing Ruby and Python quickly becomes problematic. Both frameworks try to load the BridgeSupport dylib, and this can cause errors. Some developers have posted workarounds on the web, but they tend to feel rather hackish to me. Still, I think this issue will smooth itself out with future updates.

The Bridge to OSA

We can also use Ruby and Python to communicate with scriptable applications using the Open Scripting Architecture (OSA). RubyCocoa and PyObjC already give us full access to the native Scripting Bridge, but I think this often becomes unwieldy. We end up writing Ruby (or Python) versions of Objective-C calls on AppleScript APIs.

Fortunately, each language has its own library to simplify scripting: RubyOSA for Ruby and py-applescript for Python.

Unfortunately, Leopard does not include these libraries. You need to install them on your own.

Ruby in Leopard

For the rest of this article will dig into the Ruby-specific additions to Leopard. Python has comparable features, but for simplicities sake, I will focus on what I know. Ruby comes ready for serious development. Leopard's installation includes several important libraries: rake, Mongrel, Ferret, Capistrano, sqlite3-ruby, dnssd (aka Bonjour) and Rails. Of course, given the frantic rate of Ruby development, many of these libraries have already grown long in the tooth. Still, that's not a huge concern. Leopard also includes RubyGems.

RubyGems is a command-line package manager for Ruby. It allows us to quickly and easily install and update Ruby libraries. For example, to update the current version of Rails, just type:

```
gem update --include-dependencies rails
```

However, if you're like me, the thought of wildly upgrading your system libraries makes your stomach churn. What happens if something goes wrong? Sooner or later, something always goes wrong. Won't this just screw up my system?

Well, put down that bottle of Pepto. Leopard carefully separates its pre-installed libraries from the user-installed libraries and updates. Accidentally updating to an unstable version doesn't change your original system files. Simply uninstall the offending library, and you're good to go. This also makes rolling back to factory defaults quite easy. Simply delete the user-gems folder.

Leopard keeps built-in libraries in the `/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8/` folder. The gems subfolder contains the actual libraries, while the doc subfolder contains both ri and html documentation.

When you run `gems`, it saves new libraries to the `/Library/Ruby/Gems/1.8/` folder. Again, you can find the libraries in the gems folder, while documentation is. . .wait for it. . .in doc.

Just to be complete, Leopard stashes the RubyCocoa files in a third location: `/System/Library/Frameworks/RubyCocoa.framework`

I highly recommend poking around in these directories—particularly the RubyCocoa header files. They can give you a good feel for the breadth of options available.

The Limits of RubyCocoa

Of course, there are no silver bullets, and RubyCocoa has its share of downsides.

Slow, slow, slow

As much as I love Ruby, it is a fairly slow, interpreted language. RubyCocoa code will run significantly slower than equivalent Objective-C code. Depending on the application, this may not be a problem. After all, GUI applications spend most of their time waiting on the user anyway.

Besides, if a RubyCocoa program feels slow, you can always profile it and look for bottlenecks. Once you identify likely problems, you can either redesign your code to eliminate the bottleneck, or convert it into faster, Objective-C code.

Finally, the newly released Ruby 1.9 uses a new, faster virtual machine. Unfortunately, as I write this, Ruby 1.9 only comes as a development release—it's not quite ready for prime time.

Not thread safe

Ruby 1.8 is not thread safe. You cannot call Ruby code on multiple native threads. To prevent possible problems, the bridge actually reroutes all Ruby calls from Objective-C to the application's main thread. However, as we will soon see, you can still use Ruby's threads within your Ruby code, which gives us a partial workaround. Again, the production release of Ruby 1.9 should fix this.

Xcode's debugger does not work

You cannot use Xcode's debugger on your ruby code. However, you can use Ruby's debugging tools along with new Leopard tools like DTrace and Instruments. This isn't an ideal solution, but it works.

RubyCocoa does not support Objective-C garbage collection

To me, this was probably the most disappointing limitation. Ruby itself uses garbage collection, but your Objective-C code must continue to manage its own memory. Somehow this just feels wrong.

Finding Documentation and Getting Help

Apple has included a number of documents and examples to help you get started. You can find the following articles linked off the "Introduction to Ruby and Python Programming Topics for Mac OS X" web page (http://developer.apple.com/documentation/Cocoa/Conceptual/RubyPythonCocoa):

Ruby and Python on Mac OS X

Building a RubyCocoa Application: A Tutorial

Using Scripting Bridge in PyObjC and RubyCocoa Code

The Leopard Technology Series for Developers also includes a nice introductory article at http://developer.apple.com/leopard/overview/scriptingcocoa.html.

However, if you want documentation about the frameworks that RubyCocoa supports, prepare for disappointment. You might find a promising folder at `/Developer/Documentation/RubyCocoa`. Unfortunately, this only contains a few files in Japanese. The actual RubyCocoa documentation is missing. Fortunately, we can fix this. . .more or less.

You need to download the latest RubyCocoa source release from http://rubycocoa.sourceforge.net. Untar the source files, then run the following commands:

```
ruby install.rb config
ruby install.rb doc
```

This will create ri and html documentation for most of the Cocoa libraries supported by RubyCocoa. However, the documentation has two small problems.

First, it does not cover all the libraries that RubyCocoa supports.

Second, and more importantly, the installer tends to break whenever Apple updates their reference libraries. The RubyCocoa team tries to keep up with the latest changes, but they are chasing a moving target. The 0.13.0 release will work fine for a fresh install of Xcode 3.0, but if you've updated your reference libraries, it will fail. In that case, try the latest build from the SVN trunk using the following command:

```
svn co
https://rubycocoa.svn.sourceforge.net/svnroot/rubycocoa/trunk\
/src rubycocoa
```

Don't be surprised when you see errors while parsing Apple's documentation. RubyCocoa should still create documentation for most Cocoa classes.

Alternatively, you can simply look up the Cocoa classes directly from Apple's reference library. As we will see, you can easily translate an Objective-C method into a RubyCocoa call.

Even with all the tutorials, introductory articles and reference libraries, RubyCocoa has a number of dark corners. Fortunately, you can find several other resources to help you master RubyCocoa—or at least help you ask intelligent-sounding questions.

Examples

Leopard's developer tools include 40 sample projects for RubyCocoa. You can find these in the `/Developer/Examples/Ruby/RubyCocoa` directory. These samples range from old standbys (yet another Currency Converter) to video games. Take some time to browse these projects. They can give you a real feel for using RubyCocoa effectively.

Web Sites

While a quick search on Google brings up 370,000 matches for "RubyCocoa", I highly recommend two sites: the RubyCocoa project pages at SourceForge.net (http://rubycocoa.sourceforge.net/HomePage) and RubyCocoa Resources (http://www.rubycocoa.com). Both provide a range of useful articles. The introductory topics help you get started, while the advanced topics keep you coming back for more.

The Last Resort

The RubyCocoa community has an active mailing list. In my experience, everyone is helpful and kind. But, please: don't waste their time. Try to research the issue on your own. Then, if you're still stuck, check out RubyCocoa Talk.

You can subscribe to RubyCocoa Talk at https://lists.sourceforge.net/lists/listinfo/rubycocoa-talk.

The only true graphical terminal server for Mac OS X

Aqua Connect Terminal Server is the only scalable enterprise grade solution that allows the Mac OS X platform to be deployed to multiple diverse devices, including Macs, PCs, and thin clients. Aqua Connect offers a feature set that truly simplifies OS X administration.

Mac

Universal

Aqua Connect

www.aquaconnect.net
(866) 543-AQUA (2782)

Our Project

To really understand something, sometimes you need to just jump in. Therefore, the rest of this article, will focus on building a simple RSS reader using RubyCocoa.

Why another RSS reader? Leopard already comes with built in RSS features for both Safari and Mail, not to mention many third-party applications. Still, I wanted to try something a bit messier than the typical toy project. By tackling a problem with rough edges, we get a better feel for RubyCocoa's strengths and weaknesses.

Additionally, I wanted a project that would demonstrate the following four points:

The project should use a RubyGem library.

The project should use key Cocoa technologies, like Core Data and Bindings.

The project should use RubyOSA to communicate with an existing, scriptable application.

The project should be implemented entirely in Ruby.

Our RSS reader will read and parse RSS feeds using the FeedTools gem. The application will use both Core Data and Bindings extensively. In part 2, we will send enclosures to an iTunes playlist using RubyOSA. And, except for a single Objective-C class, we will only write Ruby code.

3.859 out of 4 isn't bad.

Installing the Gems

First, a quick word of warning. Don't update RubyGems or any of your libraries just yet. As we will see, this may complicate things. Nothing we can't fix, but you might want to avoid problems when you can.

RubyGems is a powerful package manager for Ruby libraries. It is also a complex, command line tool. A full explanation is beyond the scope of this article, but the table below should get you started. For more information than you could ever possibly want, check out the RubyGem manuals at http://rubygems.org/.

Command	Result
gem	Displays basic usage information.
man gem	Displays the gems manual pages. While somewhat lacking in details, it does mention a number of other interesting utilities (like gemlock and gemwhich).
gem help commands	Displays a list of gem commands.
gem help examples	Shows examples of commonly used commands.
gem help <command name>	Displays information about a specific command.

BookEndz® Docking Stations for Apple Computers

DOCKING STATIONS FOR APPLE COMPUTERS

Convert your MacBook Pro®, MacBook®, or PowerBook®
into a desktop in seconds
without misplacing cables or damaging connectors.

- Aluminum Plate helps in cooling of notebook
- Connectors are routed to rear of Dock
- MagSafe® Housing prevents accidental disconnect

17" MacBook® Pro Docking Station

13.3" MacBook® Docking Station

- Also available in Black
- Built in DVI/VGA full size
- 5 USB 2.0 compliant ports
- Gigabit Ethernet RJ45

(4 port powered or unpowered hub, AC/DC adapter included)

15.4" MacBook® Pro Docking Station

15" PowerBook® G4 Docking Station

17" PowerBook® G4 Docking Station

12" PowerBook® Docking Station

Visit our website for latest product announcement www.BookEndzdocks.com

BookEndz®
Manufactured by Olympic Controls
1250 Crispin Drive • Elgin, Illinois 60123 • USA
Phone: 847-742-3566 • Fax: 847-742-5686 • Toll Free: 888-622-1199
E-mail: Sales@BookEndzdocks.com

gem install <gem name>	Install the named gem.
gem update —include-dependencies	Updates all installed gems.
gem update <gem name> —include-dependencies	Update just the named gem and its dependencies.
gem uninstall <gem name>	Removes the named gem.

Note: many of these commands (especially install, update and uninstall) require root access. You typically launch them as sudo commands.

Also, I deliberately left one command off the list: `gem update –system`. This updates the RubyGem system itself. Unfortunately, unlike the other gem updates, this actually changes your system files, and these changes are not easily undone.

I strongly recommend leaving this command alone. Let Apple manage the RubyGems system. As I'm writing this, they just updated RubyGems as part of the 10.5.2 release, so it should stay reasonably current. Modify the gems as much as you want, but leave the system alone.

Most of the time, you will use simple install and update commands; however, the others can come in handy when things go wrong. Updates do not always proceed as smoothly as I would like. Sometimes they leave a gem or two behaving badly. I often find that uninstalling and reinstalling the offending gem (and possibly its dependencies) sorts things out.

Now that we understand the basics of RubyGems, our first step should be the simplest. We just need to install our project's RubyGem libraries. In theory, this should only require typing the following command, entering your password when prompted.

```
sudo gem install feedtools
```

Unfortunately, life is never this easy. The FeedTools library contains the deprecated ruby-gem command. As long as you're still running the version of RubyGems that came with Leopard, you shouldn't have any problems. The library just logs a few warnings to the console. However, newer versions of RubyGems no longer recognize this command. Bottom line, if you've updated to 10.5.2, you have the new version of RubyGems, and the FeedTools library will crash.

To fix this, you simply need to edit `feed_tools.rb`. You can find this file at `/Library/Ruby/Gems/1.8/gems/feedtools-0.2.26/lib/feed_tools.rb`. Globally replace "require-gem" with "gem".

Creating the Project

Now, we can create our project. Open Xcode, and from the File menu select New Project.... In the Assistant window, scroll down and select Cocoa-Ruby Core Data Application. Click Next.

Creating our Cocoa-Ruby Core Data Application

Enter RubyRSS for the project name. Set the project directory to whatever you wish. Click Next again. Abracadabra. . .project created!

But, lets take a quick look at what Xcode has done. `MainMenu.nib` and `RubyRSS_DataModel.xcdatamodel` are standard files for any Core Data application. The first defines our user interface. The second defines our data model. We will take a closer look at both in just a second.

Open `main.m`. This is the starting point for our application. As you can see, a RubyCocoa application's main simply imports the RubyCocoa runtime, then launches `rb_main.rb` using the `RBApplicationMain()` function.

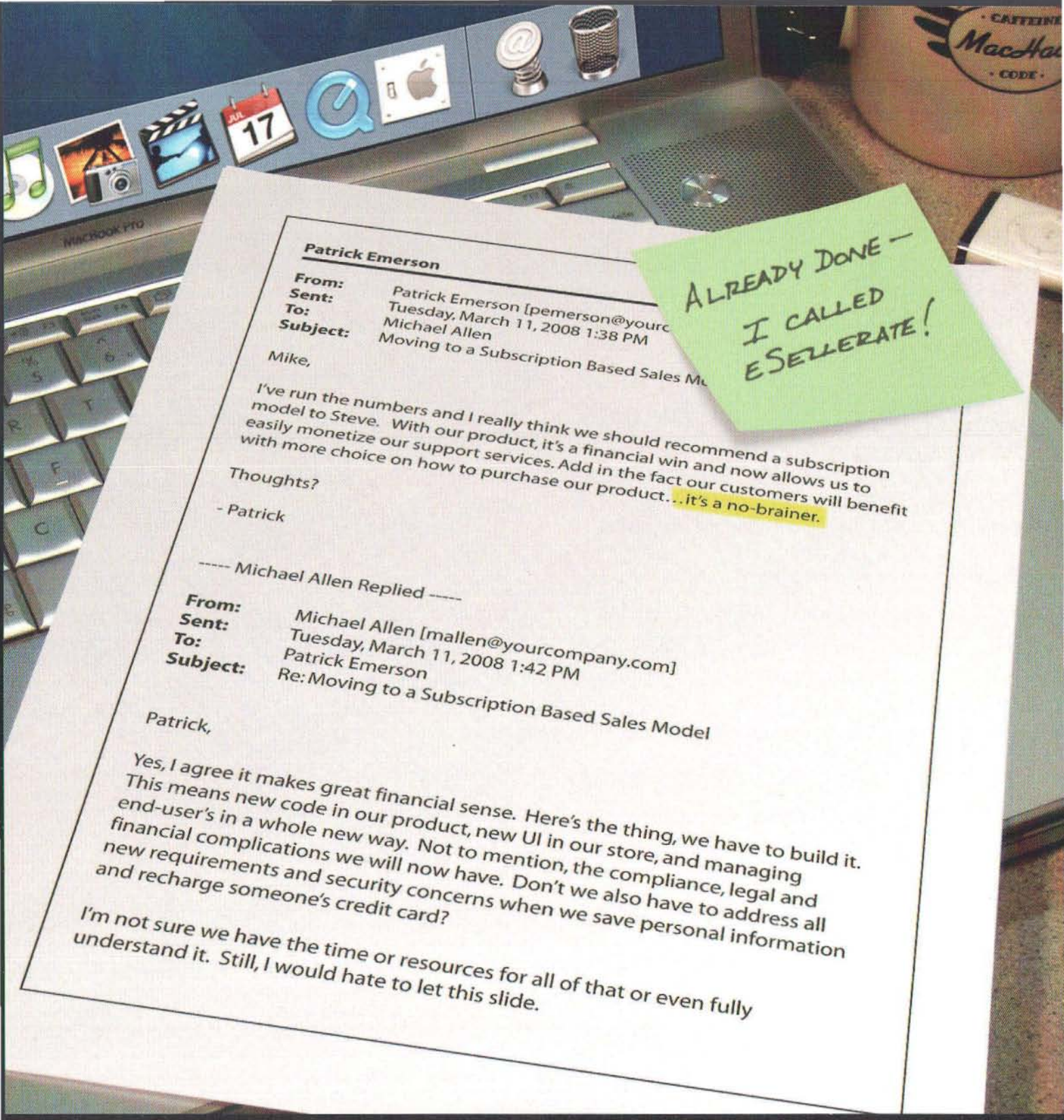
main.m

```
#import <Cocoa/Cocoa.h>
#import <RubyCocoa/RBRuntime.h>

int main(int argc, const char *argv[])
{
    return RBApplicationMain("rb_main.rb", argc, argv);
}
```

Our Ruby code really starts with `rb_main.rb`. The default implementation loads the RubyCocoa library, locates the application's resource path, then loads any files ending with .rb using Ruby's `require()` method. This creates all our Ruby classes. Once finished, `rb_main.rb` calls `NSApplicationMain()`, which initializes and runs the Cocoa application.

rb_main.rb

```
require 'osx/cocoa'

def rb_main_init

    path =
OSX::NSBundle.mainBundle.resourcePath.fileSystemRepresentation
```



eSellerate. Complete.

Begin defining your strategies at http://www.esellerate.net/mactech

eSellerate, the same team that brought you Installer Vise.

eSellerate is a registered trademark of MindVision, Inc. - a Digital River Company.

Mac Universal

```
    rbfiles = Dir.entries(path).select {|x| /\.rb\z/ =~ x}
    rbfiles -= [ File.basename(__FILE__) ]

    rbfiles.each do |path|
        require( File.basename(path) )
    end
end

if $0 == __FILE__ then
    rb_main_init
    OSX.NSApplicationMain(0, nil)
end
```

With our Ruby classes now defined, we can access them from Objective-C. Unfortunately, we cannot directly import Ruby classes into Objective-C files; however, we can indirectly access the classes by name. While we won't do this in our application, the following code snippet shows the basic technique. It creates a `MyRubyClass` object defined in a `MyRubyClass.rb` file. It then calls the object's `mySampleMethodCall()`.

Accessing Ruby from Objective-C

```
Class myRubyClass = NSClassFromString(@"MyRubyClass");
id ruby = [[myRubyClass alloc] init];
[ruby mySampleMethodCall]
```

Notice how RubyCocoa seamlessly translates objects between Ruby and Objective-C. Usually, you won't need to worry, things just work.

Sooner or later, however, you will rub up against one of the rougher edges. For example, RubyCocoa converts Ruby objects into Objective-C equivalents when possible. This means you can pass Ruby `Strings` to Objective-C methods. RubyCocoa will automatically convert them into `NSStrings`.

However, the reverse is not true. RubyCocoa will place a Ruby wrapper around Objective-C classes, and will sometimes add convenience methods (like adding `each()` to `NSString`, `NSArray` and `NSDictionary`), but it does not convert the classes.

So, if RubyCocoa calls an Objective-C method that returns a string, the Ruby code will get an NSString, not a Ruby String. A quick call to `to_s()` fixes this, but it can cause bugs if you're not careful. Also, Ruby and Objective-C sometimes have very different ideas about booleans. We'll take a closer look at that little wrinkle later.

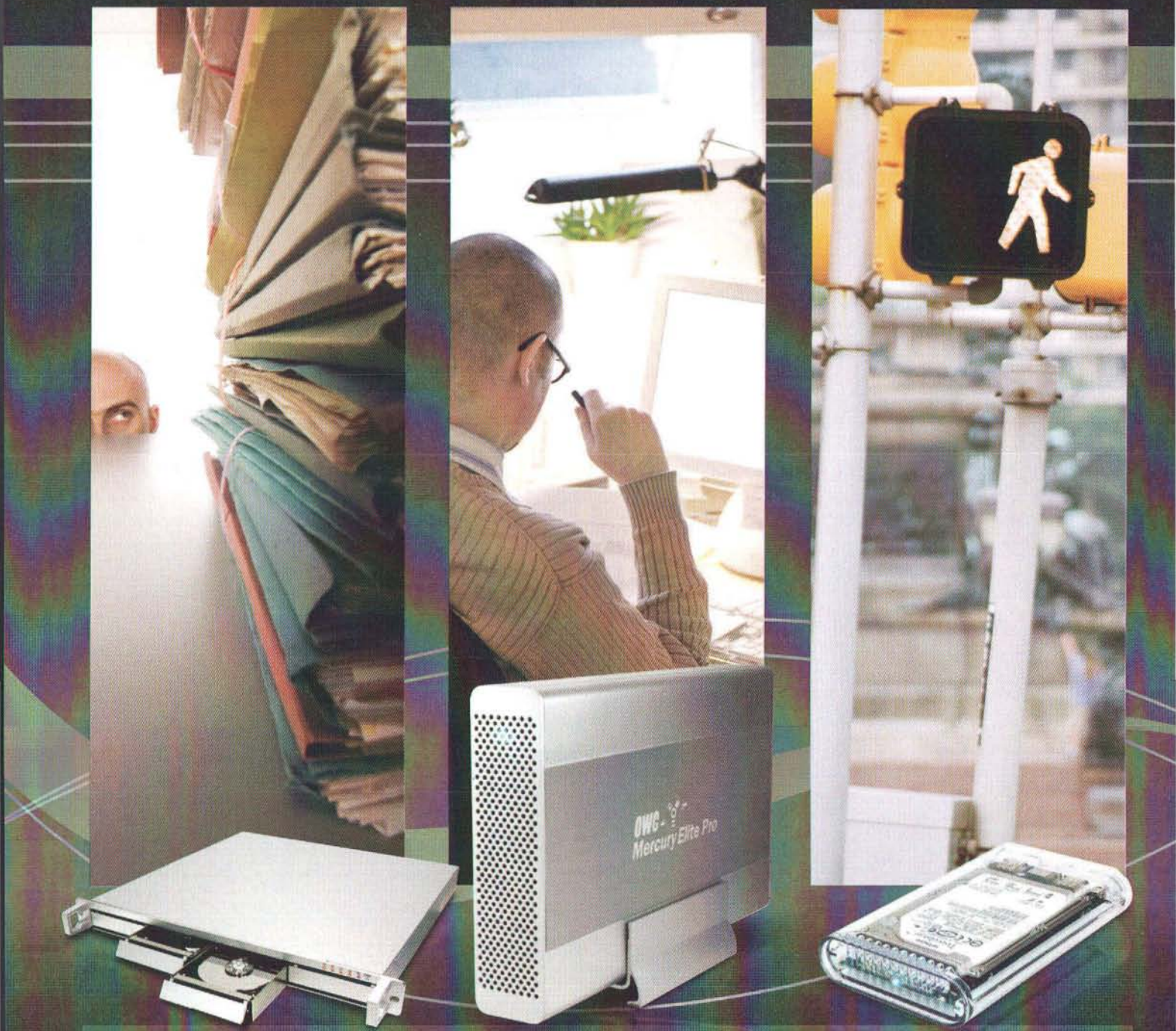
My advice, ignore object conversions until they cause problems. This is best dealt with on a case-by-case basis.

Finally `RubyRSSAppDelegate.rb` acts as a Ruby-implemented delegate for our application. Feel free to poke around this file. However, you'll find the most interesting bits at the very beginning. This class not only imports the Core Data framework, it also subclasses NSObject. This just demonstrates how easily Ruby and Objective-C code can mix.

RubyRSSAppDelegate.rb

This Ruby code imports a Cocoa framework, then subclasses an Objective-C object.

```
require 'osx/cocoa'
```



OWC® Mercury Rack Pro™ 4-Bay 1U RackMount Storage Solution

Ideal for applications requiring high data throughput, availability, and flexibility in configuration.

1.2TB to 4.0TB from $689.99

- Hardware RAID, SoftRaid, JBOD options.
- Available with eSATA, FireWire 800/400, and USB 2.0 interface support.
- User customizable configurations.
- Up to 4.0TB of performance storage.

OWC® Mercury Elite Pro™ Aluminum Storage Solution

All Mercury Elite-AL Pro models are ideally configured for Audio, Video, Digital Photography, Professional Music, Graphics, General Data and Back-Up applications.

250GB to 1.0TB from $169.99

- Rugged, machined aluminum enclosure & super quiet operation.
- Up to 32MB data buffer with data transfer rates over 150MBs!
- Ultra-protective shock isolation system.
- Premiere BONUS utility software included.
- Available with eSATA, FireWire 800/400, and USB 2.0 interface support.

OWC® Mercury On-The-Go™ Bus Powered Portable Solution

Safely and conveniently transport large amounts of data with no AC adapter needed!

60GB to 320GB from $79.99

- FW 800/400/USB2.0 to USB2.0/eSATA solutions.
- Fully suitable for audio/video applications.
- Super quiet operation with shock isolation system.
- Premiere BONUS utility software included.
- Compact 3.5"(W) x 5.5"(D) x 1"(H) size and weighs less than 11 ounces. Fits in your shirt pocket!

Other World Computing

Visit: www.owcmac.com or Call: 800.275.4576

Award-winning products: Macworld LAYERS Photoshop User

Time Machine ready means that our external hard drives keep an up-to-date copy of all of your files and document under OS X Leopard™. Go back in time to recover anything with ease.

BizRate 2007

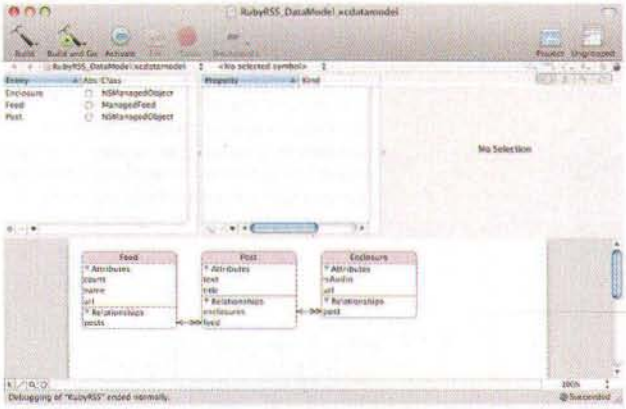
```
OSX.require_framework 'CoreData'

class AppDelegate < OSX::NSObject

...
```

Defining the Model

Building a Core Data model is beyond the scope of this article. For more information, take a look at the Core Data Tutorial video (http://developer.apple.com/cocoa/coredatatutorial) or Apple's article on creating managed object models with Xcode (http://developer.apple.com/documentation/Cocoa/Conceptual/CreatingMOMWithXcode).

For simplicity's sake, lets import our model from the online source code for this article. First, download the source code from ftp://ftp.mactech.com/src/. Delete `RubyRSS.xcdatamodel` from your project. Select Also Move to Trash when prompted. Then, select Project fi Add to Project.... In the file dialog, select RubyRSS.xcdatamodel from the source code's folder. Press Add. In the next dialog, make sure Copy items into designation group's folder (if needed) is selected. Click Add again.

Now, open `RubyRSS.xcdatamodel`, and let's poke around inside. RubyRSS uses a simple model with only three Entities: Feed, Post and Enclosure.



RubyRSS's Data Model

The Feed entities represent our RSS subscriptions. Feed has three attributes: name, url and count. It also has a too-many relationship with Post.

Post has two attributes: title and text. Post also has two relationships: one points back to Feed, while a to-many relationship points to Enclosure. So far, so good—this isn't exactly rocket science.

Finally, Enclosure has two attributes: url and isAudio. It also has a single relationship with Post.

The attributes have straightforward data types. I've listed the details below, but nothing should come as a surprise. Also, if you

look carefully at the model, you will see that I've placed some restrictions on the data. In general, I recommend making your data as restrictive as possible; however, we don't need data validation for this tutorial, so I'll let you explore it on your own.

Feed

name	String
url	String
count	Int32

Post

title	String
text	String

Enclosure

url	String
isAudio	Bool

Enclosure and Post are both `NSManagedObjects`. However, Feed's count attribute needs a bit of special attention. Count represents the number of posts associated with this feed. To get this behavior, we will need to subclass `NSManagedObjects` and override the `count()` accessor.

Now, as I said earlier, I hoped to implement everything using Ruby. This will be the one exception. Trying to write this in Ruby just creates problems; the default AppDelegate implementation automatically creates Key Value Coding (KVC) wrappers for any attributes declared in the `NSManagedObjectModel`. Since this occurs after our classes have loaded, our custom `count()` method gets clobbered.

We could fix this, but it's easier to write `ManagedFeed` in Objective-C, and I'm all about the pragmatic.

ManagedFeed.h
This is the header file for our ManagedFeed class.

```
#import <Cocoa/Cocoa.h>

@interface ManagedFeed : NSManagedObject {
}

-(int)count;

@end
```

ManagedFeed.m
This is the implementation of our ManagedFeed class.

```
#import "ManagedFeed.h"

@implementation ManagedFeed

+(NSSet*) keyPathsForValuesAffectingCount {

    NSSet *set = [super
keyPathsForValuesAffectingValueForKey:@"posts"];

    return [set setByAddingObject:@"posts"];
}
```

```
-(int)count {

    id posts = [self valueForKey:@"posts"];
    NSArray * all = [posts allObjects];

    return [all count];
}

@end
```

You can find a detailed description of the `keyPathsForValuesAffecting<key>` method in the NSKeyValueObserving protocol reference. Essentially, this method describes the dependencies for a given key. KVO uses this to determine if and when the key may have changed. In our code, `count` could change whenever the value of the `post` key changes. We could specify this by just returning a set that contains `@"post"`.

However, our implementation is a little more complicated. Apple recommends requesting an initial set of keys from the super class, then appending your own key paths to that set. In this tutorial, the call to the super class will always returns an empty set. However, this implementation protects us from future changes.

In the `count` method, we return the number of posts associated with this feed. We get a copy of the posts relationship using KVC . Then we extract an `NSArray` containing these posts. Finally, we return the number of objects in our `NSArray`.

Building the Controllers

Apple now recommends building your controllers before designing your interface. You can still create controller objects within Interface Builder and then export them back to Xcode. You can even export your controllers in Ruby and Python; however, I could not get the resulting code to run. Best to follow their advice and just write the controllers yourself.

Just like the standard Objective-C versions, our Ruby controllers combine outlets, actions and possibly a few helper functions. Outlets represent the UI elements that we will need to programmatically interact with. Actions represent UI-driven events.

Fortunately, RubyCocoa provides an `attr_accessor`-like method for defining outlets. For those not familiar with `attr_accessor`, it takes any number of symbols, and creates an instance variable for each one. `Attr_accessor` also creates the getter method `<symbol>()` and the setter method `<symbol>=()`. For example, `attr_accessor :name` creates `@name`, `name()` and `name=()`.

Similarly, `ib_outlet` takes a comma-separated list of symbols. It converts each symbol into an instance variable with the same name. A corresponding outlet will also appear in Interface Builder.

Note: you should avoid using `attr_accessor` in your RubyCocoa code. Unfortunately, `attr_accessor` does not create KVC compliant variables, so we cannot connect to them

HASP, the #1 Choice of Software Developers Worldwide*

*Source: Frost & Sullivan #N1AF-70, IDC Bulletin #34452

Finally, put a

STOP

to unauthorized software use.

HASP® HL

REINVENTING SOFTWARE PROTECTION & LICENSING

Follow the signs to HASP HL –
the market's leading software DRM solution

HASP HL provides the highest level of security for your software. Featuring easy-to-use tools, HASP HL automatically protects your software and enables new and innovative licensing options.

- ***Strong copy protection:*** *Powerful 128-bit AES encryption provides a strong locking mechanism that ensures you get paid for every copy of your software in use.*
- ***Unmatched intellectual property protection:*** *Automatically protects Windows executables, DLLs, Mac applications, and offers class-level obfuscation for .NET framework 2.0 applications. Multiple encryption layers ensure your valuable IP remains unreachable and your R&D investment is protected.*
- ***Secure & flexible licensing:*** *HASP HL enables a variety of secure and flexible licensing models and ensures end-users comply with software licensing terms.*

See for yourself why IDC ranks HASP #1.
Request a Software Developer Kit today at **www.Aladdin.com/StopPiracy_MT**

Aladdin®
SECURING THE GLOBAL VILLAGE

• NORTH AMERICA: 1-800-562-2543, 847-818-3800
• UK • GERMANY • ISRAEL • BENELUX • FRANCE • SPAIN • ITALY • BRAZIL • INDIA • CHINA • JAPAN

www.Aladdin.com/HASP

using Bindings. The getter works fine, but Cocoa expects a `set<Symbol>()` setter (`setName()` in our example).

Fortunately, RubyCocoa provides `kvc_accessor`. `Kvc_accessor` works identically to `attr_accessor`, but creates KVC compliant methods.

RubyCocoa also simplifies declaring KVC dependencies. The `kvc_depends_on()` method takes two parameters: an array of symbols representing the dependencies, and a single symbol representing the calculated attribute.

Basically, this method replaces Objective-C's `keyPathsForValuesAffecting<key>()`. Take a look at our `ManagedFeed.m` file again. The `keyPathsForValuesAffectingCount` method defines count's dependency upon posts. In Ruby, we could replace that method with a single line:

```
kvc_depends_on([:posts], :count)
```

Finally, RubyCocoa elegantly handles actions. Simply define a method with a single parameter, usually named `sender`. After the method, add a call to `ib_action()` passing in the method's name as a symbol.

Sample RubyCocoa Action

```
def myAction
    ...
end
ib_action :myAction
```

Now, the Rubyists out there have undoubtedly noticed that the RubyCocoa formatting looks a bit odd. Most of this creeps in when we translate Objective-C syntax into Ruby.

Objective-C's syntax uses both named arguments and colons—neither of which translates nicely. Therefore, when referring to an Objective-C method, concatenate all the pieces of its signature, and replace the colons with underscores.

```
[canvas print: text withFontColor: red];
```

becomes

```
canvas.print_withFontColor_(text, red)
```

As a bit of syntactic sugar, RubyCocoa allows you to drop the final underscore. So, `print_withFontColor_()` becomes `print_withFontColor()`. Note: the *Ruby and Python Programming Topics for Mac OS X* article claims that this option is disabled by default. This is not true. In most cases, you can use the two variants interchangeably. The exceptions, however, can cause real pain.

When Objective-C calls a Ruby method that overrides an Objective-C method (Ah, yes. She knows that I know that she knows that I know. . . .), RubyCocoa looks for the method signature without the trailing underscore. So, just to prevent possible problems, I recommend universally dropping the last underscore.

For consistency, I've tried to use camel case for actions (likeThis). Pure-ruby helper functions have the more-traditional underscore names (like_this).

OK, enough babbling. Let's look at the code. We will have two windows in our UI, the main window, and a dialog for adding new feeds Let's create a controller for each: `MainController.rb` and `AddFeedController.rb` respectively.

MainController.rb

This class acts as the controller for our main window. It responds to all the main window's actions, and makes changes to the data model. It will also coordinate with both the FeedTools and RubyOSA libraries when necessary.

```
require 'osx/cocoa'

# Controller for the Main window.
class MainController < OSX::NSObject

    ib_outlet :feeds, :posts, :enclosures, :web_view,
:posts_table,
        :enclosures_table, :progress, :app_delegate

    # accessor for the current Feed collection.
    def feeds
        return @feeds.arrangedObjects
    end

    # Add remaining methods here
end
```

Here, we're building a subclass of `NSObject`. We start by declaring a slew of outlets for Interface Builder. The `feeds()` accessor returns an array containing all our Feed entities. This represents all currently subscribed feeds.

The next two methods override `NSObject` methods. The Cocoa framework will automatically call these.

NSObject Methods

```
# Initializes the Main Window after it is loaded from the
NIB.
def awakeFromNib

    @progress.setDisplayedWhenStopped(false)

    @posts.addObserver_forKeyPath_options_context(self,
"selection",
        0, nil)
end


# This listener method will be called whenever the Post
Table's selection
# changes. It updates the HTML in the web view.
def observeValueForKeyPath_ofObject_change_context( key_path,
object,
    change, context)

    set_html if @posts.isEqual(object)
end
```

The framework calls our `awakeFromNib()` method after all objects have been loaded from the nib file, and once all outlets are set. We can use this method to perform any additional initialization. In our case, we make the `NSProgressIndicator` invisible when not in use. We also

force our controller to listen for any changes to the `@posts` selection.

The framework now calls `observeValueForKeyPath_ofObject_change_context()` whenever `@posts`'s selection changes. We simply verify that we're receiving an update from `@posts`, then call the `set_html()` helper method.

Note: As I mentioned earlier, you must drop the final underscore from this method's name. Otherwise, Key Value Observing (KVO) cannot find our implementation.

Next, we declare two actions: `sendToItunesAction()` and `refreshFeedsAction()`. Currently, they just print a message to the console.

Actions

```
# This action sends the currently selected Enclosure to
iTunes.
def sendToItunesAction(sender)
    puts "Send to iTunes"
end

ib_action :sendToItunesAction


# This action refreshes all the feeds.
def refreshFeedsAction(sender)
        puts "Refresh Feeds"
end

ib_action :refreshFeedsAction
```

`Add_feed()` adds a new Feed entity to the managed object context. It then fills in the feeds attributes. Notice that it leaves the posts relationship blank. We don't have any posts yet.

add_feed()

```
# Adds a new feed to the Managed Object Context
def add_feed(name, url)

    moc =  @app_delegate.managedObjectContext

    new_feed = OSX::NSEntityDescription\

.insertNewObjectForEntityForName_inManagedObjectContext("Feed"
,
            moc);

    new_feed.setValue_forKey(name, "name")
    new_feed.setValue_forKey(url, "url")

end
```

Finally, `set_html()` gets the text from our currently selected post. We then display this text in our web view.

set_html()

```
private

# Helper Function: Updates the HTML displayed by the Web View
to the text
# of the currently selected post.
def set_html

    index = @posts_table.selectedRow
    frame = @web_view.mainFrame
    # If nothing is selected, just return.
    if index < 0 then
        frame.loadHTMLString_baseURL("", nil)
    else
        post = @posts.arrangedObjects[index]
        frame.loadHTMLString_baseURL(post.text, nil)
    end

end
```

Our second controller is even simpler. This controller has only two outlets, plus two KVC-compliant properties, and a third virtual property.

The `sheet` outlet provides access to the Add Feed dialog sheet, while the `window_controller` provides a link back to our main controller.

The `name` and `url` properties hold (not surprisingly) the name and URL of the new feed.

Finally, the virtual property, `valid_feed`, returns true if the feed has a valid name and URL. Obviously, `valid_feed` depends upon the `name` and `url` properties. Key Value Observing will call our `valid-feed` accessor whenever either of the dependent variables changes.

AddFeedController.rb

```
require 'osx/cocoa'

# AddFeedController acts as the controller for the Add Feed
sheet.
class AddFeedController < OSX::NSObject

    ib_outlet :sheet, :window_controller
    kvc_accessor :name, :url
    kvc_depends_on([:name, :url], :valid_feed)


    # Add methods here

end
```

The `open_dialog()` method opens the Add Feed dialog. We declare this as an action, so that we can link it to a button on the main window.

open_dialog()

```
# This action opens the Add Feed sheet.
def open_dialog(sender)

    OSX::NSApp.beginSheet_modalForWindow_modalDelegate_\
        didEndSelector_contextInfo(    @sheet,

@main_window,
                                                    self,
                                                    nil,
                                                    nil)

end

ib_action :open_dialog
```

Next, we add our `add_feed()` action. We will link this action to the Add button on the Add Feed dialog sheet. This method simply converts the feed name and URL into Ruby Strings, then delegates back to the main window controller's `add_feed()` method. Finally, it closes the Add Feed sheet.

add_feed()

```
# The add_feed action grabs the name and url from the Add
Feed sheet.
# adds the new feed to the Managed Object Context, then
closes the sheet.
def add_feed(sender)

    feed_name = @name.to_s
    feed_url = @url.to_s

    @window_controller.add_feed(feed_name, feed_url)

    close_dialog
end

ib_action :add_feed
```

The `cancel()` action simply closes the Add Feed dialog sheet. We will link this action to the Cancel button on the Add Feed sheet.

cancel()

```
# The cancel action closes the sheet without adding a new
feed.
def cancel(sender)
        close_dialog
end

ib_action :cancel
```

The `valid_feed()` method uses Ruby's regular expressions to filter out invalid entries. Basically, the feed name must contain at least one non-whitespace character, while the URL must start with "feed://", then contain one or more characters, a period, and end with one or more characters. The URL cannot have any white space.

Note: While Ruby has explicit true and false values, it also treats all nil values as false, and all non-nil values as true. This means, the result of ANDing together two regular expressions is either `nil` or the String matched by the second regular expression. While Ruby will correctly interpret this as a boolean value, when we pass it to the Cocoa framework, we get the following exception:

```
AddFeedController#rbSetValue_forKey: OSX::OCException:
NSInternalInconsistencyException - Cannot create BOOL from
object <RBObject: 0x1437bdd0> of class RBObject
```

To prevent this, we explicitly convert our result to a boolean value.

valid_feed()

```
# valid_feed returns true if the Add Feed sheet's current
name and url
# values are valid. This method can be monitored using KVO.
def valid_feed

    name = @name.to_s
    url = @url.to_s

    result = name.match('\S+') &&
url.match('^feed://\S+\.\S+$')
```

```
    # explicitly convert to booleans.
    return ! result.nil?
end
```

Finally, the private helper function, `close_dialog()` clears the Text Fields and closes the dialog.

close_dialog()

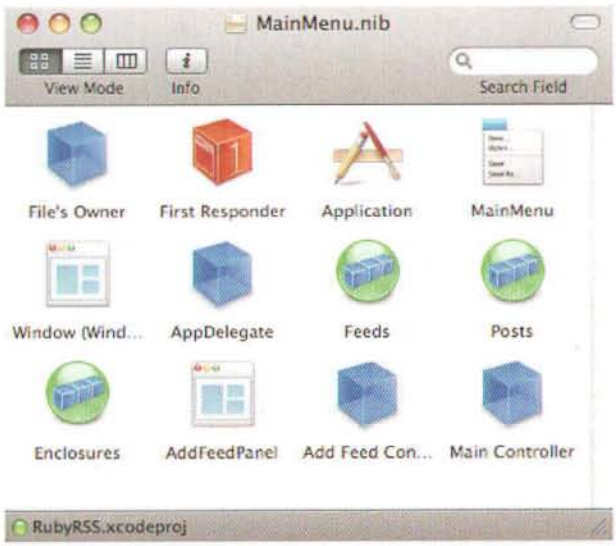
```
private

# Helper Function: close_dialog clears the Add Feed's
# text boxes, then closes the sheet.
def close_dialog

    setName("")
    setUrl("")

    @sheet.orderOut(self)
    OSX::NSApp.endSheet(@sheet)
end
```

Building the User Interface

Building the user interface is also beyond the scope of this article. Simply copy MainMenu.nib from the online source code.

Our user interface consists of the main window and the Add Feed panel. We have three array controllers. The first contains all of our `Feed` entities. The second contains all `Post` entities associated with the currently selected `Feed`. The third contains all `Enclosure` entities associated with our currently selected `Post`. Bindings automatically maintain these relationships, requiring no code on our part.

Finally, we have the `Add Feed` and the `Main` controllers defined in the previous section.



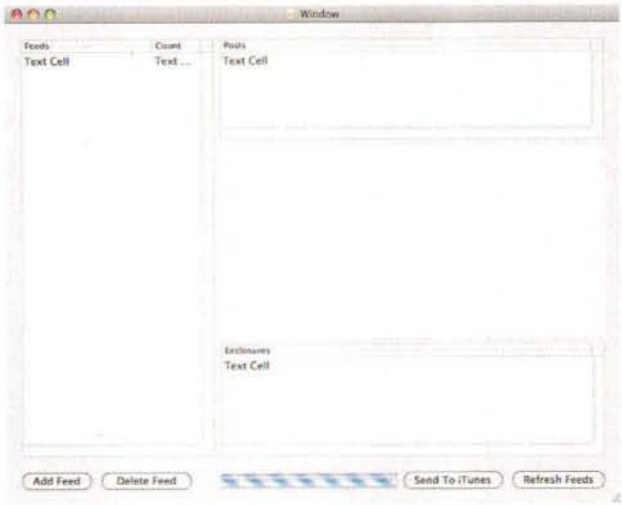
RubyRSS's Arrays and Controllers

The Main window consists of three Table Views. The first contains the names and post counts from the `Feeds` array. The second displays titles from the `Posts` array. The final table contains URLs from the `Enclosures` array. Again, we set all of these values using Bindings. Of these, only the feed names are editable.

The Main window also has a Web View. This contains the text for the currently selected post; however, unlike the Table Views, we cannot set the Web View's content using Bindings. Instead, we actually have to write code.

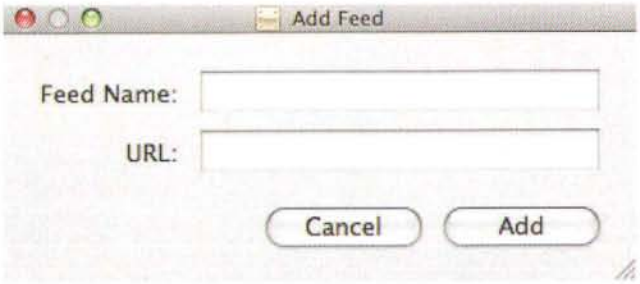
The good news is, you've already written this code. Look back at our main controller. Remember, how it receives notifications whenever the `posts'` selection changes? It then fires the `set_html()` helper function. That's the code we need. We use KVO to automatically synchronize our web view with the current selection. Basically, we're recreating the code that Bindings normally gives us for free.

Finally, our main window has four Buttons: one adds a new feed, one deletes the selected feed, one sends the selected enclosure to iTunes, and the last one refreshes all our feeds. Since some of these operations can take a long time, we also have a Progress Indicator.



RubyRSS's Main Window

Even simpler, the Add Feed panel has two Text Fields: one for the Feed's name and one for the URL. Each Text Field has a corresponding Label. Finally, we have an Add Button and a Cancel Button.



Add Feed Window

The connections between our UI elements and the controllers' outlets and actions should seem straightforward enough. I won't go into the details here, but I encourage you to open up the nib in Interface Builder and get a feel for the wiring.

One last quick step. Since our user interface uses a Web View, we need to add the `WebKit.framework` to our project. Right click on the Frameworks folder in the Groups & Files tree. Select Add fi Existing Frameworks.... In the File dialog, find and select the `WebKit.framework` and select Add. In the next panel, just select Add again.

You can now compile and launch the application. Of course, it won't do much yet. We can add new feeds, but we cannot actually read or parse them. All the basic RubyCocoa code works, but we still need to add support for FeedTools and RubyOSA.

Parsing the Feeds

The FeedTools library provides code for parsing, generating and auto discovery of RSS, atom and cdf feeds. We're only using a fraction of its abilities. If you want to know more, check out the web page for FeedTools and its sister project FeedUpdater (http://sporkmonger.com/projects/feedtools/).

Let's make a new class to handle the interactions between FeedTools and our data model. Create a new Ruby class named `FeedReader.rb`.

FeedReader.rb

```
require 'rubygems'
require 'feed_tools'
require 'osx/cocoa'
OSX.require_framework 'CoreData'

# FeedReader class uses feed_tools to download and parse all
the feeds.
# then adds new posts and enclosures to the Managed Object
Context.
class FeedReader

    # insert methods here
end
```

FeedReader starts by loading the required libraries. Obviously we need FeedTools and RubyCocoa. The RubyGems library lets us access any gem-installed libraries; therefore, we need to load RubyGems before loading FeedTools. Most interestingly, the OSX.require_framework method lets us load Cocoa frameworks. In this case, FeedReader needs Core Data.

initialize()

```
# Default Constructor
def initialize(main_controller, moc)
    @main_controller = main_controller
    @moc = moc
end
```

`Initialize()` allows us to construct new FeedReader objects. It takes two arguments: a reference to the main window controller, and a reference to our managed object context.

refresh()

```
# Gets the current list of feeds. Downloads all Feeds and
adds new Posts
# and Enclosures to the Managed Object Context.
def refresh
    feed_entries = @main_controller.feeds
```

```
    feed_entries.each {|data_feed| update(data_feed)}
end
```

FeedReader only exposes a single method to the outside. `Refresh()` iterates over all the feeds, passing each one to the `update() helper function`.

update()

```
private

# Helper Function: updates a single Feed.
def update(data_feed)
    feed = FeedTools::Feed.open(data_feed.url)
    posts = feed.entries
    posts.each{|post| add_post(post, data_feed)}
end
```

`Update()` extracts the list of available posts for each feed. It iterates over the list of posts, calling `add_post()` for each one.

add_post()

```
# Helper Function: adds new posts to the given feed.
def add_post(post, data_feed)
    title = post.title
    text = post.summary

    # check to see if this already exists...
    return if post_exists?(title, text)

    # now make a new entity
    data_post = OSX::NSEntityDescription\

.insertNewObjectForEntityForName_inManagedObjectContext("Post"
,
            @moc);

    data_post.setValue_forKey(title, "title")
    data_post.setValue_forKey(text, "text")
    data_post.setValue_forKey(data_feed, "feed")

    enclosures = post.enclosures

    enclosures.each do |enclosure|
        add_enclosure(enclosure, data_post)}
    end
end
```

`Add_post()` extracts the post's title and text. It then calls `post_exists?()`, checking if any posts in the managed object context already have a matching title and text. If the post doesn't already exist, `add_post()` adds a new `Post` Entity. It then fills in the entity's attributes and sets the feed relationship. Since we're using bi-directional relationships, Core Data automatically adds this post to its `Feed`. Finally, `add_post()` iterates over the post's list of enclosures, calling `add_enclosure()` for each one.

post_exists?()

```
# Helper Function: determine if a post exists with the given
title and
# text.
def post_exists?(title, text)

    request = OSX::NSFetchRequest.alloc.init

    description = OSX::NSEntityDescription\
        .entityForName_inManagedObjectContext('Post', @moc)

    request.setEntity(description)

    predicate = OSX::NSPredicate.predicateWithFormat(
        "title like %@ AND text like %@", title, text)

    request.setPredicate(predicate)

    error = nil

    count = @moc.countForFetchRequest_error(request, error)

    # if we have an error, assume the post doesn't exist.
    if !error.nil?
        puts "*** Error ***"
        puts error
        return false
    end

    count > 0
end
```

`Post_exists?()` builds an `NSFetchRequest` for all `Post` entities whose title and text mach the given arguments. We then use `countForFetchRequest_error_()` to count the number of matching Posts. For any number greater than zero, we return true. Otherwise, we return false. Note: we log any errors, but simply assume no matches are found.

add_enclosure()

```
# Helper Function: adds a single Enclosure to the given Post.
def add_enclosure(enclosure, data_post)

    url = enclosure.url
    isAudio = enclosure.audio?

    new_enclosure = OSX::NSEntityDescription.\

insertNewObjectForEntityForName_inManagedObjectContext(
            "Enclosure", @moc);

    new_enclosure.setValue_forKey(url, "url")
    new_enclosure.setValue_forKey(isAudio, "isAudio")
    new_enclosure.setValue_forKey(data_post, "post")
end
```

Finally, `add_enclosure()` adds a new `Enclosure` entity to the managed object context. It then fills the attributes, and sets the post relationship. Again, we have a bi-directional relationship, so Core Data automatically adds this `Enclosure` to the `Post`'s enclosures relationship.

Now we just need to make our MainController aware of the FeedReader. Add the following line to MainController's `awakeFromNib()` method:

```
@feed_reader = FeedReader.new(self,
@app_delegate.managedObjectContext)
```

This creates a new FeedReader object. We just need to call `@feed_reader.refresh()` whenever the user presses the Refresh Feeds button. However, this operation can take a while, especially when you subscribe to a lot of feeds. We don't want our UI to freeze up. Also, we would like to let the user know that something is actually happening. So, let's have FeedReader refresh the feeds in a second thread, and turn on the progress bar.

Unfortunately, Ruby 1.8 is not thread safe. We cannot call Ruby code from a second (or third, or fourth...) Objective-C thread. However, we can use Ruby's internal threads. Ruby uses

a green threading model. Basically, as far as the hardware knows, Ruby runs in a single thread, but the Ruby interpreter can time slice between several green threads.

Ruby threads have some advantages and some disadvantages over processor threads. A full discussion is beyond the scope of this article, but — bottom line — they work fine for our purposes. The user interface will not freeze up while we refresh the feeds.

Simply replace `refreshFeedsAction()` with the following code.

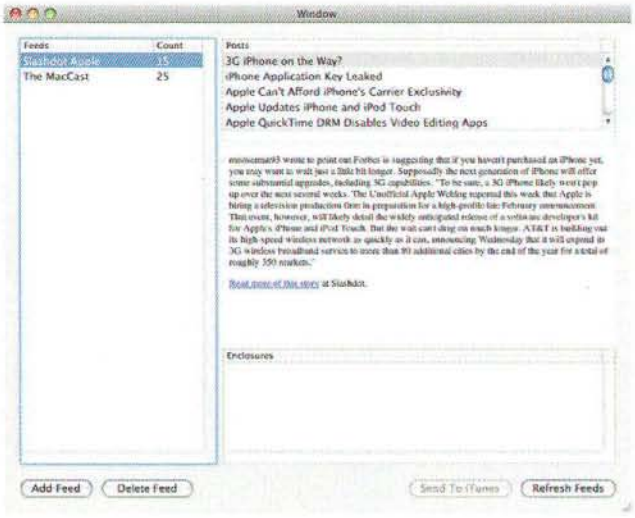
new refreshFeedsAction()

```
# This action refreshes all the feeds.
def refreshFeedsAction(sender)
    Thread.new do
        begin
            @progress.startAnimation(self)
            @feed_reader.refresh
            @progress.stopAnimation(self)
        rescue Exception => e
            puts e.message
            OSX::NSApp.stop(self)
        end
    end
end

ib_action :refreshFeedsAction
```

The `begin. . .rescue. . .end` blocks handle any exceptions thrown in our worker thread. If an error occurs, it executes the rescue block, which logs the error and quits the application.

I'm a firm believer in failing fast. We don't want our code to lumber ahead in an unknown state. At least during development, go ahead and force the application to stop as soon as an error occurs.

That's it. Open up the RSS reader and add a few feeds. Click the Refresh Feeds button, and watch the posts roll in. Quit the application, and then launch it again. Core Data automatically saves all our data.



The Complete RubyRSS

This almost looks like a real application. Almost. It still needs a lot of work. For example, the tables remain completely unsorted. Ideally, users will want to filter them as well. By default we should probably filter out any posts that the user has already read. Searching would also be nice.

From a software engineering standpoint, we're playing fast and loose with our threads here. The user can perform any number of bad actions (like quitting the application) while the worker thread is still running. We should probably address that.

But, you have to admit, we squeezed a ton of functionality out of a few hundred lines of code. Ruby + Core Data + Bindings makes a high-octane combination.

Next time, we will look at using RubyOSA to send enclosures to iTunes. We will also take a look at RubyCocoa's debugging options, and look at a few other cool tricks as well.

About The Author

Rich Warren lives in Honolulu, Hawaii with his wife, Mika, daughter, Haruko, and his son, Kai. He is a software engineer, freelance writer and part time graduate student. When not playing on the beach, he is probably writing, coding or doing research on his MacBook Pro. You can reach Rich at rikiwarren@mac.com, or check out his blog at http://freelancemadscience.blogspot.com/

Grokking OS X's Undo Support

How to effectively use the Undo Manager built into Cocoa

by Marcus S. Zarra

Introduction

One of the great benefits of working with Cocoa is that the APIs give the developer numerous features "for free". One of those features is undo support. Any Cocoa application, without any work from the developer, will automatically get undo support at some basic level. In this article I walk through exactly how Undo support works in Cocoa and how you as the developer can take some control over undo to provide the functionality you are looking for.

Overview

Undo is one of those features that is rarely thought about. Most developers do not check it off as a feature in their application and most users do not immediately look for it when reviewing a new application. However, it is a feature that if not present when you need it, it is sorely missed. Fortunately, Apple has recognized this and included undo support for the developer so that, in most cases, we do not need to think about it.

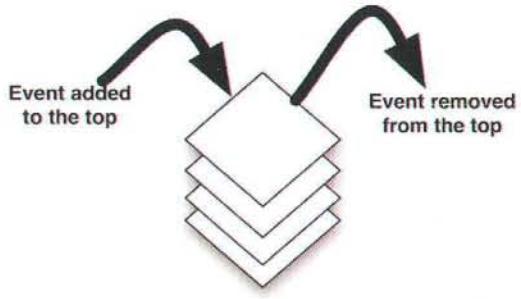
Unfortunately, when your application strays from the beaten path then you need to roll up your sleeves and make sure that undo support works exactly as you expect it to. As it is said, the devil is in the details, and having inconsistent undo support is worse than having none at all.

What is Undo

Apple defines undo as:

An undo operation is a method for reverting a change to an object, along with the arguments needed to revert the change (for example, its state before the change)

Undo is performed on a stack. What this means to those not familiar with the term is that each action, or event, that is "undoable" is added to the top of the stack and therefore can be removed from the stack. When you remove an item from the stack it is also removed from the top and thus reversing the order it was added. For example, if you were editing a paragraph of text in a Cocoa NSTextArea and you typed "Hello World", that text would be added to the stack as an undoable event. If you then selected "Hello" and hit ?B, the text would become bold and the action of bolding that selection of text would be added to the stack.



The reason that the "stack" aspect of this is important is when you want to undo something. When you choose undo then the first time is removed from the top of the stack (in this example, the bolding of "Hello") and undone. If the items were processed in a FIFO (First In, First Out) order then the text would be deleted instead, which is definitely not what the user would expect to happen.

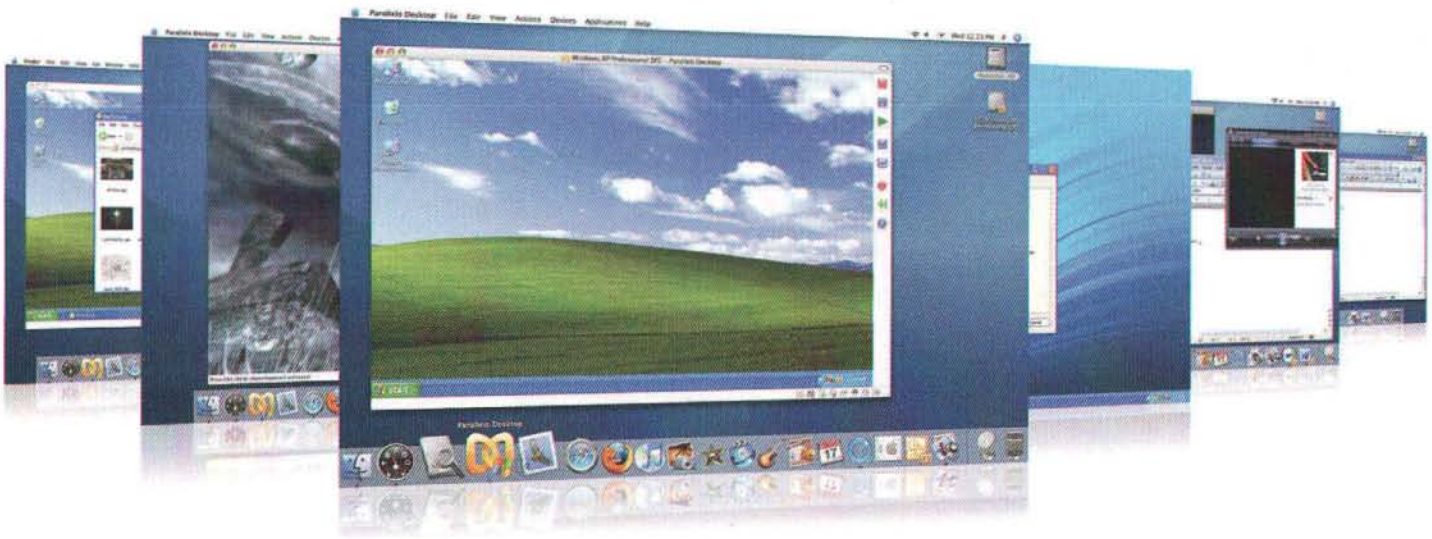
The Undo Stack

What exactly is the stack? Probably the easiest way to describe the stack is to describe how to put things onto it. As an example, lets say that we have a value that we want to be undoable and therefore we want to add any changes to that value to the stack. To do that we would perform the following:

```
-(void)setName:(NSString*)newName;;
{
    [[self undoManager] registerUndoWithTarget:self
        selector:@selector(setName:)
        object:[self name]];
    [newName retain];
    [name release];
    name = newName;
}
```

Plays well with others.

Run Windows and Mac OS X at the same time - without rebooting.



With Parallels® Desktop 3.0 for Mac, you can:

- Automatically open Windows files with Mac software and Mac files with Windows software
- Bulletproof your virtual machine from Windows system crashes and malware
- Run selected Windows-only 3D games and applications with 3D graphics support
- Browse through Windows files and folders without launching Windows
- Move your existing Windows PC to a Mac without losing data or reinstalling any software
- Completely share files and folders between Windows and Mac

Parallels™

Parallels® Desktop 3.0 for Mac provides even more ground breaking features and capabilities than previous versions. Already The #1 Selling Mac System Utility according to NPD Techworld (09/06 – 12/07) and recently named an InfoWorld Magazine "2008 Technology of the Year", Parallels® Desktop is trusted by over 800,000 users world wide.

Want better Windows and Mac integration? Want to run the hottest PC games and graphics software? Worried about security and system mishaps? Parallels® Desktop 3.0 for Mac includes 50+ new features and enhancements, including a number of integration features and virtual machine utilities unavailable anywhere else.

To discover why Parallels® Desktop is the leading desktop virtualization solution for running Windows on a Mac, visit us online or call us at 1 (425) 282-6405.

PC WORLD 100 BEST PRODUCTS of 2007

Macworld BEST OF SHOW 2007

Download a trial today at www.parallels.com/download

Website – www.parallels.com Email – sales@parallels.com Phone – 1 (425) 282-6405

For those familiar with Cocoa/Objective-C, this is a fairly standard setter call to set the attribute name. The change from the normal is the additional call to the NSUndoManager. Note that in this example I am passing it a reference to the object to call the method on (self), the method name to call (setName:) and the *previous* value of that attribute.

Internally, the NSUndoManager, is going to remember these values that you are passing in and retain them as an undo event. That event will then be stored on a stack (presumably in an NSArray) to be recalled at a later time. Therefore the stack is really an array of structures/objects that reference a target, a selector and another object. One thing that is interesting in this design is that the object (the previous value of name) is actually *retained* by the undo event. This guarantees that the value will still be around if the undo event is ever invoked.

When the user of the application invokes the undo command, the last item added to the NSUndoManager is retrieved and then the selector is called upon the target with the object that is being passed in, thereby reversing the event. Of note: When an event is removed from the undo stack, it is automatically added to the redo stack. This allows a user to undo and redo their actions as needed.

Grouping Undo

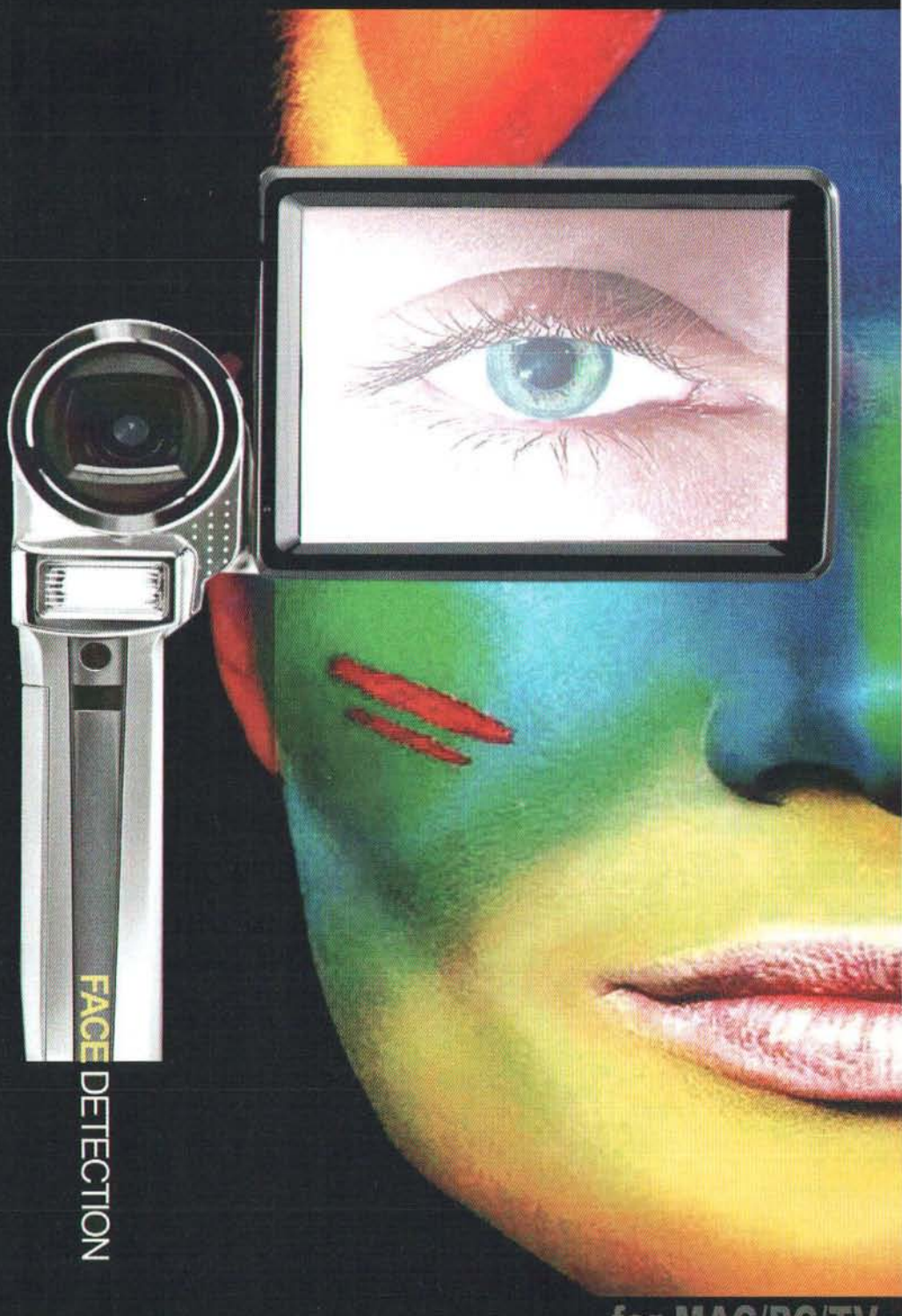
In addition to being able to undo and redo individual events such as the setter above, it is possible to group individual events together so that they are undone and redone as a single operation. An example of this would be an extension of the example above. For instance, lets imagine that the setName: method above actually breaks the string apart into a first name and last name. And instead of having an undo registration at the setName, we want to have the first and last name register events, then grouping would come into play when the full name is set.

```
-(void)setName:(NSString*)name;
{
    [[self undoManager] beginUndoGrouping];
    NSArray *words = [name componentsSeparatedByString@" "];
    [self setFirstName:[words objectAtIndex:0];
    [self setLastName:[words objectAtIndex:1];
    [[self undoManager] endUndoGrouping];
}

- (void)setFirstName:(NSString*)name
{
    [[self undoManager] registerUndoWithTarget:self
        selector:@selector(setFirstName:)
        object:[self firstName]];
    [name retain];
    [firstName release];
    firstName = name;
}

- (void)setLastName:(NSString*)name
{
    [[self undoManager] registerUndoWithTarget:self
        selector:@selector(setLastName:)
        object:[self lastName]];
    [name retain];
    [lastName release];
    lastName = name;
}
```

As can be seen in the listing, the individual parts of the name have stored their changes in the NSUndoManager's stack

supacam.com

The ONLY camera that streams DVD Quality Video across the internet



for MAC/PC/TV

supastream.com

"The full VGA streaming gives a picture **that brings Video sharing to a new height"**

Professor Norman Medoff author Portable Video ENG

but the setName: method does not store its changes in the undo stack. Instead, it starts a group, sets the individual components and then ends the grouping. What this does is still the NSUndoManager that all undo events it receives between the begin and end are to be performed together. By using grouping in an example like this you do not risk polluting the stack with a setName:, setFirstName: and setLastName: call and thereby causing an issue when the user attempts to undo the action.

Naming the event

In addition to being able to control the undo/redo events, it is also possible to name these events. That name is then used by the undo and redo menu items to help explain to the user exactly what they will be undoing and redoing.

```
- (void)setFirstName:(NSString*)name
{
    [[self undoManager] registerUndoWithTarget:self
        selector:@selector(setFirstName:)
        object:[self firstName]];
    [[self undoManager] setActionName:
        NSLocalizedString(@"First Name",
        @"First Name undo action")];
    [name retain];
    [firstName release];
    firstName = name;
}
```

The only change from the previous example is the addition of [undoManager setActionName:]. This call instructs the NSUndoManager to attach the passed in string as the name of the *previous* event. In this example, it attaches the name of "First Name" to the event of setFirstName:. If the user were to look in the edit menu after changing the first name they would see something like "Undo First Name". As you can see, we localized the action name so that we can easily go back and localize the name of the action at a later time.

It should be noted that groupings can also be named. If an event and a grouping are both named then the grouping name will win out. For example, if our setName: method also set the ActionName to "Name" then when the setName: method is called the edit menu would display "Undo Name" and not "Undo First Name" or "Undo Last Name".

Levels of Undo

As can be expected, since the NSUndoManager actually retains the objects being passed around it is quite possible to begin to have a memory issue. This is especially true if the object of the function call is in itself quite large. One option to help contain this is to limit the number or "levels" of undo available to the application. This is controlled by a simple call to:

```
[[self undoManager] setLevelsOfUndo:10];
```

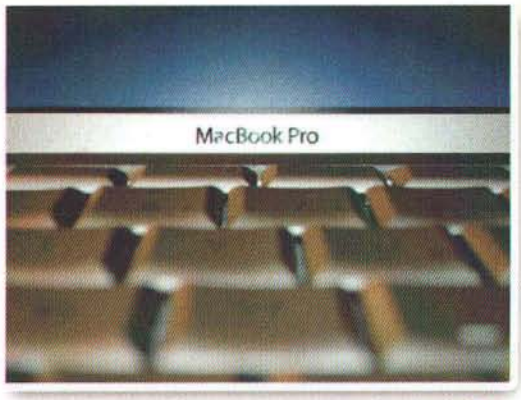
which will limit the application to remembering only the last 10 undo events.

Blocking Undo Support

As you can probably imagine, since undo support is managed as such a low level (all the way down in the setters),

Short Term Computer Needs?

- Latest Apple Technology
- Legacy Macs
- Windows Computers



- Rent One or Hundreds
- Audiovisual Gear
- Projectors, Plasmas, LCDs
- B&W & Color Copiers

Mac Rentals
Event Technology Services

- Name Brands
- Nationwide Delivery
- Professional Setup
- Onsite Technicians
- Same Day Service

- Compatibility & Compliance Testing
- Short Term Projects, Production Needs
- Trade Shows, Conventions, Corporate Events
- Trainings Sessions, Classrooms
- Fortune 100 and Educational P.O.'s Accepted!

Toll Free

800-756-6227

Outside US/Canada: 858-454-8535

For Online Quote Request, Visit:

www.madcomputing.com

Largest Apple Rental Inventory in the U.S.!

it is possible to build up a large undo stack just by populating an object from another source. Fortunately, there is a way to avoid this. For example, imagine you are populating our example object from an XML feed. Naturally, you do not want to start building up the undo stack while setting the values from the xml document but you do want to add to the stack when a user changes something. How can you instruct your application to tell the difference?

The secret is to pause the undo registration while loading your document and then resuming it after your load is complete. See the following example:

```
- (MyObject*)buildMyObjectFromXML:(NSXMLDocument*)doc
{
    NSString *tempFirstName = ...;
    NSString *tempLastName = ...;
    NSUndoManager *undo = [self undoManager];
    [undo disableUndoRegistration];
    MyObject *object = [[MyObject alloc] init];
    [object setFirstName:tempFirstName];
    [object setLastName:tempLastName];
    [undo enableUndoRegistration];
    return [object autorelease];
}
```

In this example, we instruct the NSUndoManager to stop registering events and then we build the data object and set its attributes. The data object itself has no knowledge of whether or not the NSUndoManager is disabled (although we could check if needed for some reason) and continues to register events. The NSUndoManager receives these events and since it is disabled simply throws them away. When the object has been completely loaded we then turn the registration back on so that any events afterwards will be registered properly.

Initializing the NSUndoManager

In all of these examples, I have just been calling [self undoManager] to get a reference to the NSUndoManager. In an actual program you should have very few NSUndoManagers. Each NSUndoManager should be in a logical location. For instance, if you have a document style application then each document should have its own NSUndoManager. Otherwise you probably only want one NSUndoManager for the entire application.

To initialize an NSUndoManager, you simply call:

```
NSUndoManager *undoManager = [[NSUndoManger alloc] init];
```

Naturally, you are going to want to hold onto a reference to this object so that you can make it available to other parts of your application.

Getting the UI to use your NSUndoManager

If you are going to build and/or use your own NSUndoManager then you will want to let the user interface access that same NSUndoManager to avoid having multiple stacks trying to act on the same data. Fortunately, all views get their NSUndoManager from their window and the window gets its NSUndoManager from its delegate. Therefore if you have set up your controller (or another object) as the delegate for your window, add the following method call:

```
- (NSUndoManager *)windowWillReturnUndoManager:(NSWindow
*)window
{
    return [self undoManager];
}
```

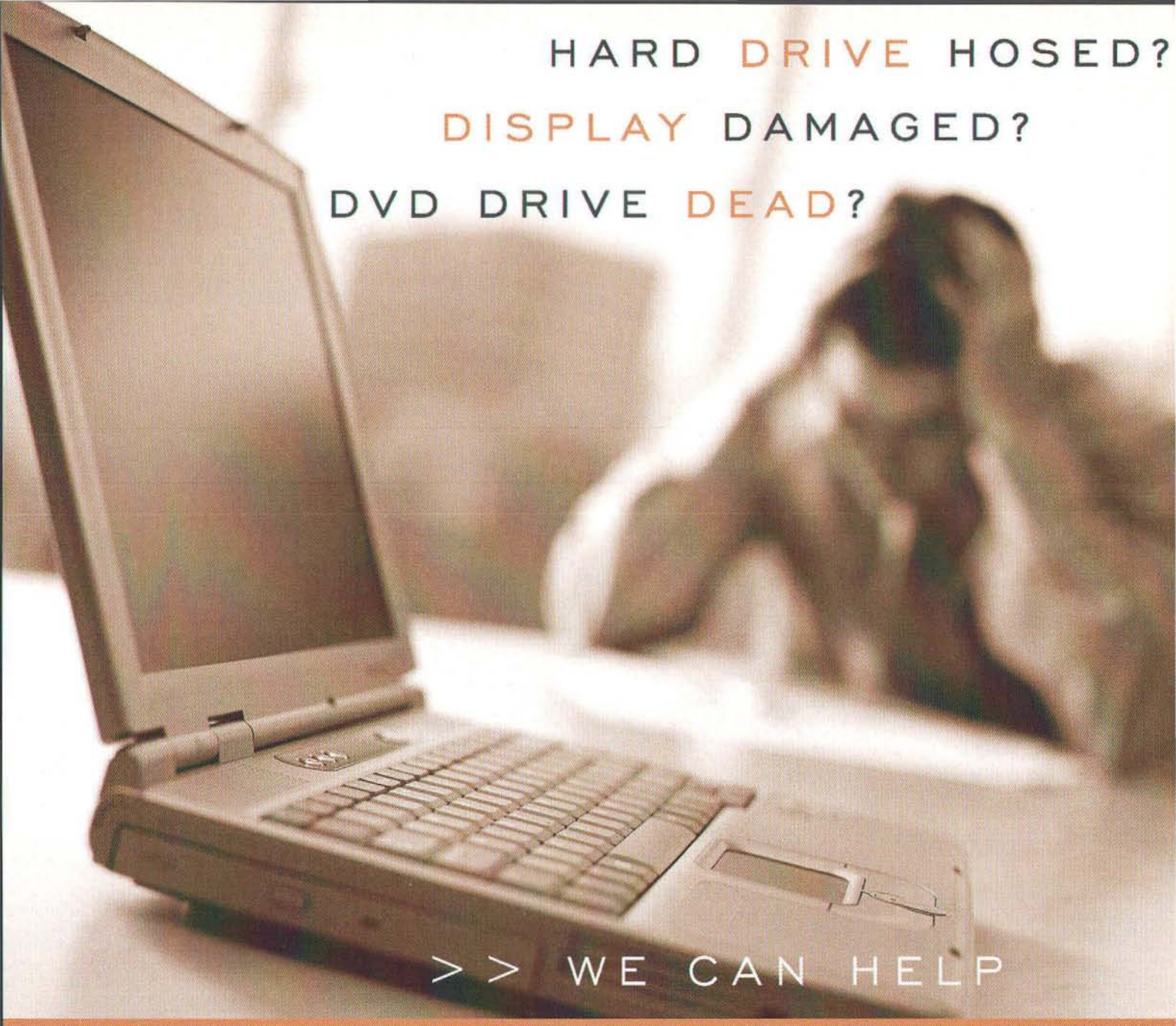
The window will then call this method to get its NSUndoManager. If you do not supply a delegate then the window will initialize its own NSUndoManager.

Conclusion

Once the curtain has been pulled back you can see that the NSUndoManager is in fact a simple array containing structures with two objects and a selector. That is all there is to Undo. However that simplicity belies the power of the design. The application of that simple data structure allows for the sense of wonder and awe that is an application with a properly implemented undo subsystem.

About The Author

Marcus S. Zarra is the owner of Zarra Studios, based out of Colorado Springs, Colorado. He has been developing Cocoa software since 2003, Java software since 1996, and has been in the industry since 1985. Currently Marcus is producing software for OS X. In addition to writing software, he assists other developers by blogging about development and supplying code samples.



REPLACEMENT PARTS • UPGRADES • REPAIRS FOR YOUR MAC LAPTOP

- FREE ONLINE DO-IT-YOURSELF GUIDES
- SEND-IN REPAIR SERVICE
- BATTERY REPLACEMENTS FROM $ 94.95
- POWER ADAPTERS FROM $ 24.95

VISIT US ON THE WEB>>
OR CALL 866.726.3342

PowerbookMedic.com

Your Mac ~ Our Patient

ww.PowerbookMedic.com
5 Sparkman Dr. Ste. 1620
untsville, AL 35816
l/Fax: 866.726.3342

by Dave Dribin

Nice and Gooey

Writing your first Cocoa GUI interface

Graphical User Interfaces

This is the tenth article in *The Road to Code*, and we have yet to write an application with a graphic user interface, or GUI. We've stuck to writing command line applications, which may seem strange, given that Mac OS X is considered one of the best GUI environments. But, as with learning many complex topics, you've got to learn the basics before moving on to the more advanced topics. Well, you'll be happy to know that we finally reached the point where we can start writing GUI applications for Mac OS X in Objective-C. Congratulations!

Leopard Only

A word of warning before getting too deep: now that Mac OS 10.5 (Leopard) is available, I will be writing all examples in Xcode 3 and Interface Builder 3, the new integrated development environment, or IDE, that comes with Leopard. This will not only affect Interface Builder screenshots and instructions, but the code will also be targeted to Leopard, as well. I will be making liberal use of Objective-C 2.0 properties, fast enumeration, and garbage collection. These new features can simplify code and make it more readable, so I think all new code going forward should take advantage of them. Since we haven't gone over these topics outside of the *Leopard Detour* article, I'll be going over them in more detail as we come across them. If you don't have Leopard or want to target previous version of Mac OS X, the simple examples we'll be going over for the time being can be built with minor modifications on Xcode 2.4.1 and Interface Builder 2.5. I'll try to point out any potential areas that are not backward compatible.

Creating a New Project

With the technicalities out of the way, let's get started. Open up Xcode 3 and select the File > New Project... menu. Choose Cocoa Application as in Figure 1.

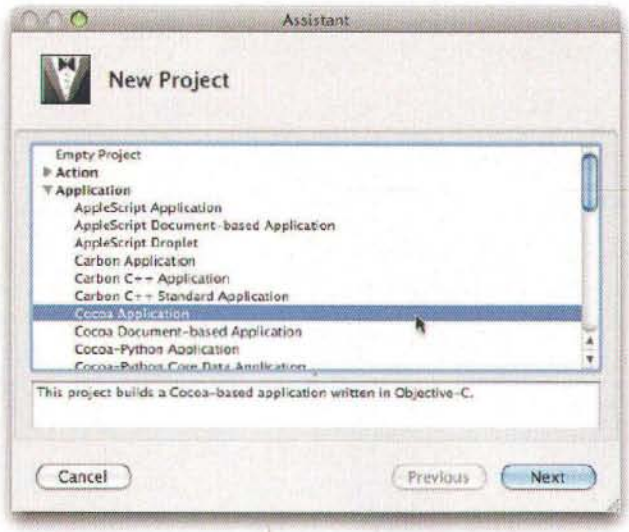


Figure 1: New Cocoa application project

Name the project Hello World and click Finish. So far, this is the same as creating a new Foundation Tool application as we have been doing previously. However, this project has a different layout than command line tools. If you open some of the disclosure triangles in the Groups & Files section on the left, you should see a list of files similar to those shown in Figure 2.

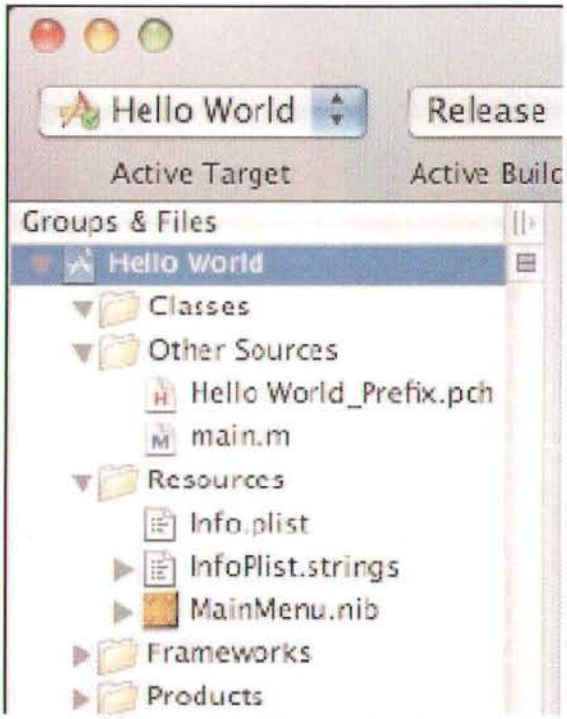


Figure 2: Cocoa Application default files

Revolutionize your Lifecycle Management Strategy with LANrev!

True Cross-Platform Lifecycle Management has arrived!
Do you need to manage PCs with a Mac? Macs with a PC?
LANrev is the first client management solution
with true cross platform functionality
for today's heterogeneous client environments.

Disk Imaging
Asset Inventory
Software Distribution
License Management
Patch Management
Remote Management
Role-Based Administration
Change and Configuration Management
LANrev TheftTrack Client Recovery

LANrev is a scalable multi-platform solution, which can operate with equal efficiency on a single server deployment in the small to medium enterprise, or in a distributed architecture, managing thousands of desktops in a large enterprise.

LANrev's modern, modular platform allows seamless integration into other desktop management tools such as Microsoft's SMS, enabling your organization to leverage previous investments while maximizing current and future efficiencies through the use of innovative new technology such as LANrev. The best part? You can take advantage of your current hardware infrastructure. Unlike the other guys, LANrev does not have massive, dedicated server requirements and you won't need a team of overpaid consultants to deploy and use LANrev.

See for yourself how you can increase your IT efficiency and performance while lowering your cost by using LANrev in your computing environment. Schedule a free web demo and trial license by calling (214) 459-0136 or visiting www.lanrev.com.

Full Mac OS X 10.5 Leopard Support!

LANrev

Mac Universal for windows

You'll notice it creates more than the single `.m` file that we have seen thus far. It creates a `main.m` source file and a few other resources. We'll talk about all these files in a second, but it turns out that this program is ready to run. So let's not waste anymore time and see what Xcode gives us by default. If you select Build and Go from the toolbar or Run > Run from the menu bar, this will start our new application. You will get a new empty window, as in Figure 3.

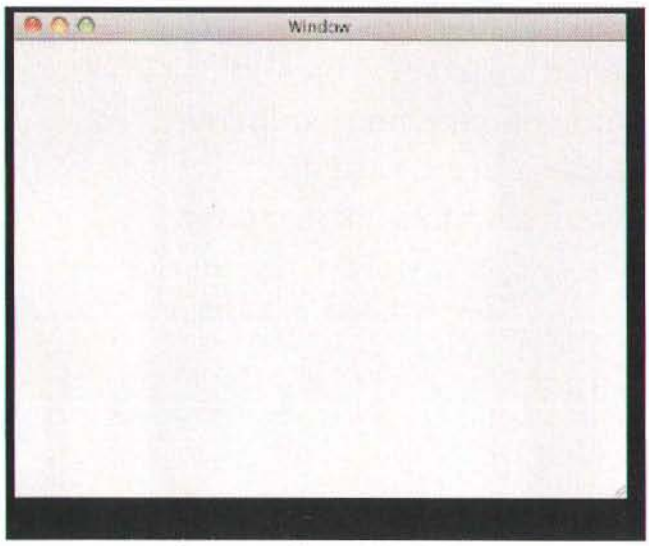


Figure 3: Hello World window

That's pretty basic, but it's a fully functional Cocoa GUI application. You even get a menu bar, as shown in Figure 4. In fact, you need to quit our application to return to Xcode so choose Hello World > Quit NewApplication to do this now.

Figure 4: Hello World menu bar

You many have noticed the "NewApplication" text in the Hello World menu, as shown in Figure 5. Xcode doesn't setup the menus with the name of our application automatically. But don't fret, we can edit the menus to our heart's content, and we'll change this text in a bit.

Figure 5: Hello World quit menu

Before going further, let's take a quick look at the source code that Xcode generates for us. There's only one source file and that's `main.m`. Select that to see what's inside. If you're expecting to see a bunch of code that sets up the menus and creates a window, you'll be sorely disappointed. Our `main` function is quite small. In fact it's only a single line:

Listing 1: main function for Cocoa applications

```
#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}
```

That's all the code running this application. The only other source file is the `Hello World_Prefix.pch` file, and this file is only used to help speed up compile times. It's called a *prefix header* and can generally be ignored. So where *do* the window and menus come from?

Interface Builder

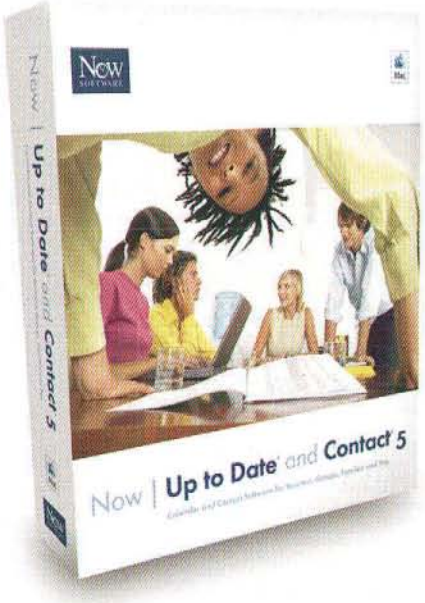
If you look in the Resources group, you'll notice a file named `MainMenu.nib`. A nib file contains all the GUI components of a Cocoa application in a sort of freeze-dried state. These nib files cannot be edited in Xcode, however. There is a new application called Interface Builder for this. Double click on `MainMenu.nib` and Xcode will automatically open this file for you in Interface Builder. As a historic note, the nib file extension stands for NeXT Interface Builder and once again shows the NeXT lineage of Mac OS X.

Once this file is opened up in Interface Builder, you will be presented with a myriad of new windows. I've shown a bird's-eye view of all the windows in Figure 6.

Now **scheduling** and **contact management** for your entire organization.

Now | **Up to Date**® and **Contact**® **5**

Calendar and Contact Software for Business, Groups, Families and You.



Is this project on schedule? When are you available to meet about the systems upgrade? Where are all the field techs today? When was the last time anyone talked to our biggest customer?

Virtually all groups live (or die) by their abilities to meet deadlines and keep track of their customers, prospects, and vendors. Few small companies or even departments of big companies have the tools they need.

Now Up-to-Date & Contact might just be the calendar and contact software for you. It's time-tested and used by more Mac-based companies than any other solution. And it's cross-platform–available for your PC users, too. It's easy to install and manage and simple for your employees to understand and use.

Using Now Up-to-Date & Contact you can schedule meetings for multiple users, view multiple, simultaneous calendars, and reserve rooms and resources. You can share contact information about your customers, prospects and vendors. And using our free server software you can set it up in minutes and share with users in the office or from anywhere with an internet connection.

Call us now at 866-527-0556 or email us at mactech@nowsoftware.com and we'll send you our free evaluation kit, including the book that will make it all easy, "Take Control of Now Up-to-Date & Contact" from Take Control books!

NOW SOFTWARE

Phone: 866-527-0556

Web: www.nowsoftware.com

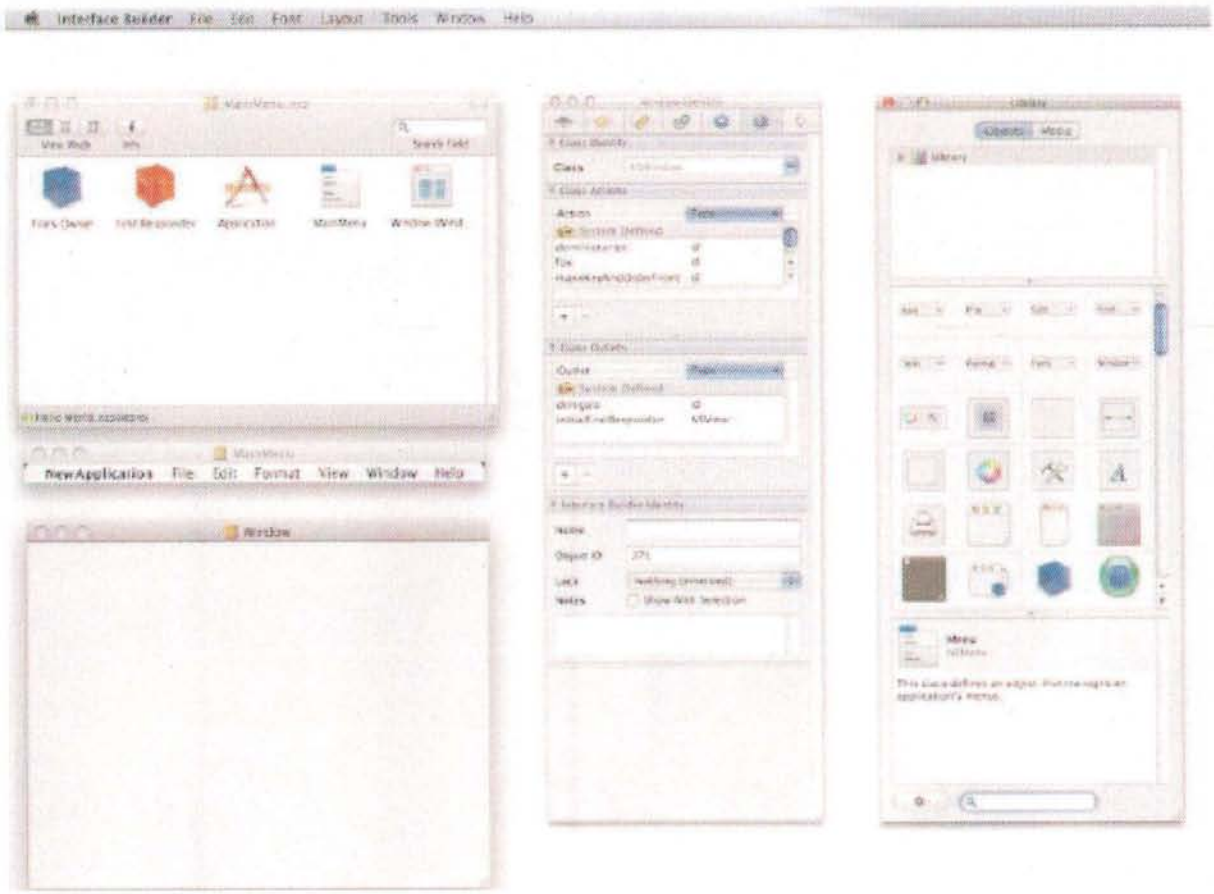


Figure 6: Interface Builder windows

The two panel windows on the right are not part of our nib. The first one with the Window Identity title is the Inspector panel. This is how we change various attributes of our GUI components. The Library panel contains all the "parts" that we *can* add to our nib. The MainMenu.nib window contains all the "parts" that currently *are* part of our nib. Among these components you'll see MainMenu and Window. These correspond to the freeze-dried main menu and window of the Hello World application. The other two windows correspond to editable copies of the main menu and window of our application, which is how we change them.

Let's start by adding a text label to our window. You'll need to find the Label component in the Library. The easiest way to do this is to type "label" into the search field. Once you find it, drag it from the Library panel onto your Window, as shown in Figure 7.

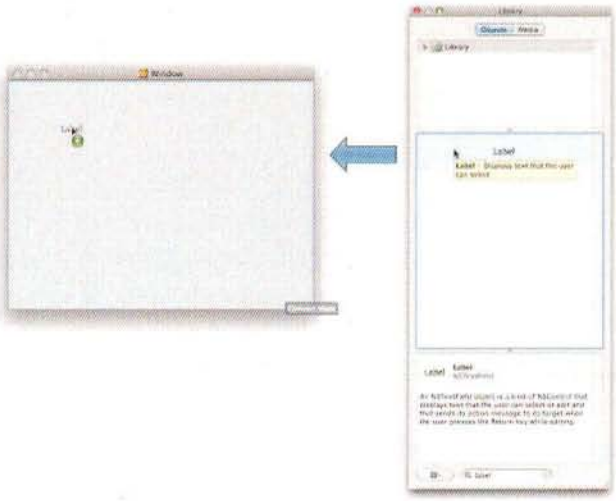


Figure 7: Adding a label to our window

Now that you have the label placed in your window, you can change the text. Double click on it and change the text to "Hello World". You can also move the label anywhere you like, just by dragging. If you switch back to Xcode and run the application, the window should have our new label in it, as shown in Figure 8.

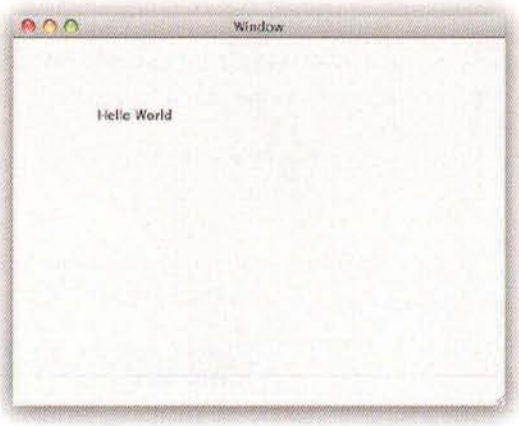


Figure 8: Running Hello World with the new label

See how easy that was? We were able to add a label to our application without even writing a single line of code. Interface Builder is a powerful GUI editor. As you start writing more complex GUI applications, you will be spending a lot of time in Interface Builder, as well as Xcode.

Changing the Menus

Before we start adding some custom code to our application, let's fix up the application name in the menus. Go back to Interface Builder and click in the window that looks like a menu bar. You should be able to open the menus and see all the menu items, just

like a normal menu bar. However, you can now double click on the menus or menu items to change the text. For example, to change the text "About NewApplication" to "About Hello World," open up the **NewApplication** menu and double click on the **About NewApplication** menu item. You should now be editing the text, as shown in Figure 9. Change the text appropriately and hit **Return**.

Figure 9: Editing a menu

Now do the same with the **Hide** and **Quit** menu items, as well as the **NewApplication** menu in the menu bar. The final result should look like **Figure 10**.

Figure 10: Edited menu

If you now go back to Xcode and run the application, you should see our edited menus.

Actions

While you can do a lot using Interface Builder alone, you do end up having to write code that interacts with the user interface. Interface Builder provides ways to hook up parts of the user interface with code, as well. Let's add a button to our window, but first delete the label we added. Click on the label and delete it by pressing the **Delete** key. Now, we need to find a button in the Library. If you type "button" into the Library's search field, you will see a bunch of different button styles to choose from. The standard Cocoa button is called a **Push Button** and is the first one in the list, as shown in Figure 11. Drag the push button to your main window and change its text to "Press Me," as shown in Figure 12.

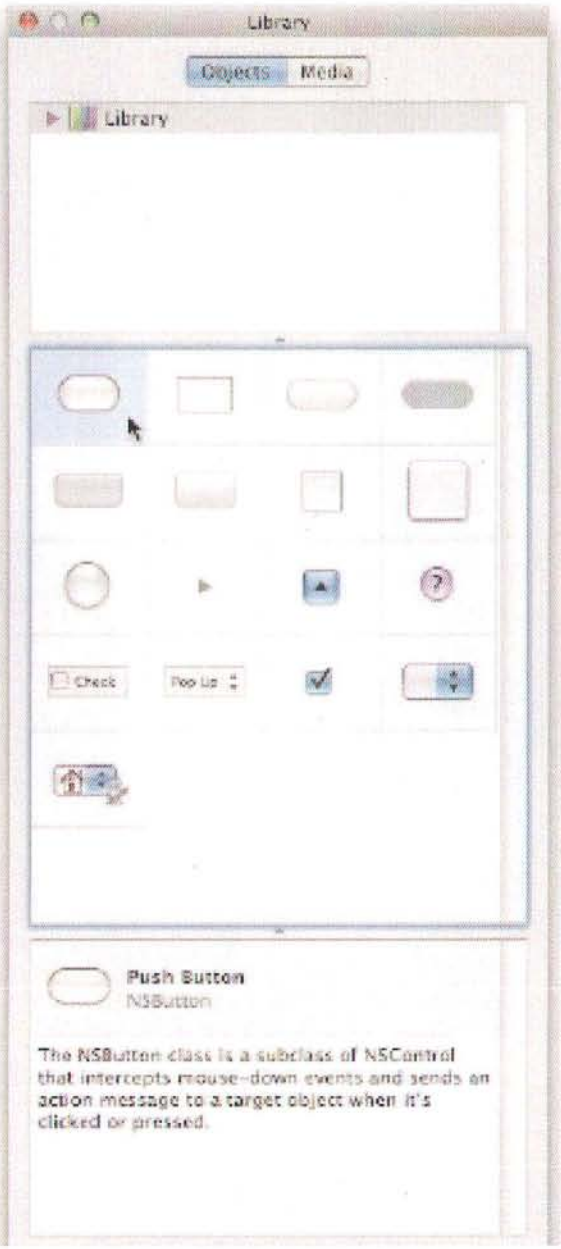


Figure 11: Buttons in the library

The Missing Sync™

Reliable Handheld Synchronization for Mac

Thanks to **The Missing Sync**, Mac users are no longer second-class citizens when it comes to keeping their mobile devices in sync. Whether it's a **Windows Mobile 6** device, a **BlackBerry** or even a **Palm** Treo, there's a Missing Sync product made to connect and synchronize that device with Mac OS X.

Syncs Contacts, Calendars, Tasks

- Supports Address Book, iCal, Microsoft Entourage 2004
- Outstanding field support, even syncs Address Book contact photos
- Supports calendar event reminders and detached events
- Sync Services-savvy for syncing with third-party apps

Complete Notes Syncronization

- Includes Mark/Space Notebook for Mac to create, edit, categorize, sort and search through notes
- Also supports Microsoft Entourage 2004 and Bare Bones Yojimbo
- Full support for Sync Services and .Mac syncing between Macs

iPhoto and iTunes Integration

- Imports photos and videos from mobile devices into iPhoto albums or folders in the Finder
- Resizes and downloads selected iPhoto albums to the device for handheld viewing and slideshows
- Downloads DRM-free music and podcasts for mobile playback

The Missing Sync for BlackBerry, Palm OS or Windows Mobile is available in single-user licenses for $39.95 or in multi-user packs for any size organization, The Missing Sync family of products provide a Mac-centric synchronization solution second to none.

Visit www.markspace.com/reliable today to see how easy it is to sync the latest smartphones and mobile devices with the Mac.

mark/space

Figure 12: Press Me button

You can run the application as is, and you can even press the button, but nothing will happen. We need to hook up the button to some code so we can do something when the user presses it.

Let's create a new class in Xcode and name it `HelloWorldController`. Xcode will create the empty header and the implementation files, as usual. We're going to create a method that gets called when the button is pressed. Make the header and source file match Listing 2 and Listing 3, respectively, and save both files.

Listing 2: HelloWorldController.h

```
#import <Cocoa/Cocoa.h>

@interface HelloWorldController : NSObject
{
}

- (IBAction) pressMe: (id) sender;

@end
```

Listing 3: HelloWorldController.m

```
#import "HelloWorldController.h"

@implementation HelloWorldController

- (IBAction) pressMe: (id) sender;
{
    NSLog(@"Button pressed");
}

@end
```

This class contains one method, `pressMe:`, that prints a message to the console when called. This method has a very special signature. It returns a type called `IBAction`, and it takes one argument of type `id`. The `IBAction` return type is really the same as `void`, meaning there is no return value, with the exception that it marks this method as an *action method* for Interface Builder. We haven't yet covered the `id` type, but it's not really important for this discussion. We'll tackle `id` in a future article. What's important is that action methods take a single `id` argument.

Marking a method as an action method means you can access this method from Interface Builder. Before we can use it, we need to create an instance of `HelloWorldController` that Interface Builder can see. Go back to Interface Builder and search for "nsobject" to find the **Object** component in the Library. Drag this over to the **MainMenu.nib** window, as shown in Figure 13.

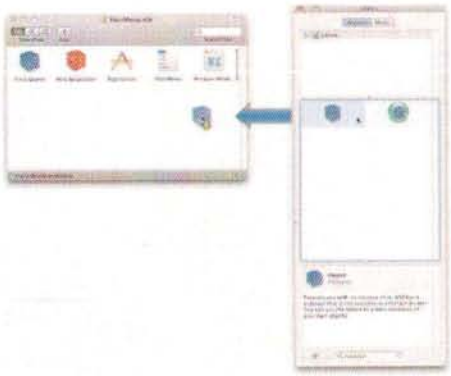


Figure 13: Adding an NSObject to your nib

Now go to the **Inspector** panel, and click on the **Identity** tab; it's the sixth tab from the left with the little "i" in a circle icon. Change the class to **HelloWorldController** and hit **Return**. The panel should now look like Figure 14. If Interface Builder does not allow you to choose **HelloWorldController**, be sure you've saved your files in Xcode.

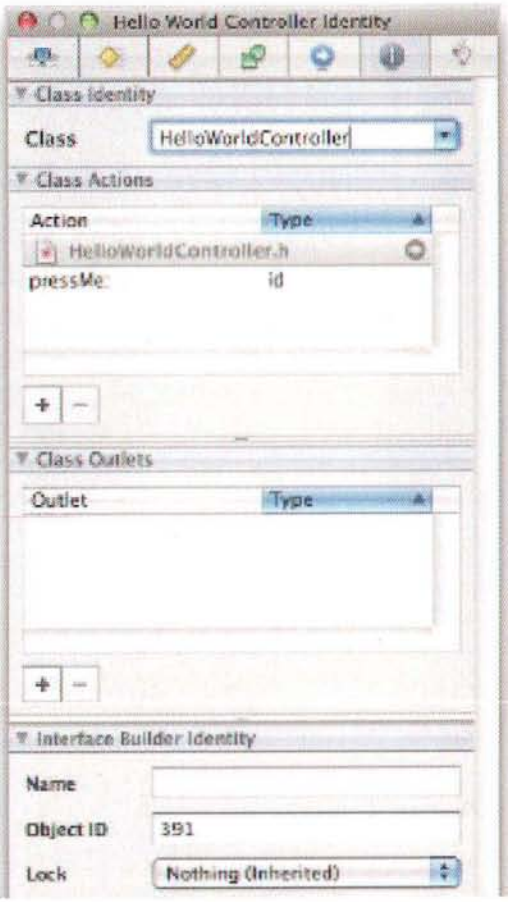


Figure 14: Setting the class of an NSObject

You'll notice that once the class is set to `HelloWorldController`, the `pressMe:` method also shows up

in the Class Actions section. If you don't see this action then double check your code to make sure you spelled everything correctly. By adding our object to the nib, we've created a freeze-dried instance of our class. This means that one instance of our class will automatically be created when the application starts.

With our object in place and our action visible to Interface Builder, we can hook up the action to our button. You do this by control dragging *from* the button *to* the Hello World Controller. By "control dragging" I mean you must press and hold the Control key *while* clicking and dragging. This action may be a bit strange, but you will use it a lot in Interface Builder. You should see a blue line follow your mouse as you drag. After you release the mouse button on top of our object, you'll get a popup window of possible actions you can connect the button to, as shown in Figure 15. Click on `pressMe:` and Interface Builder will flash a bit to confirm that it hooked up the action.

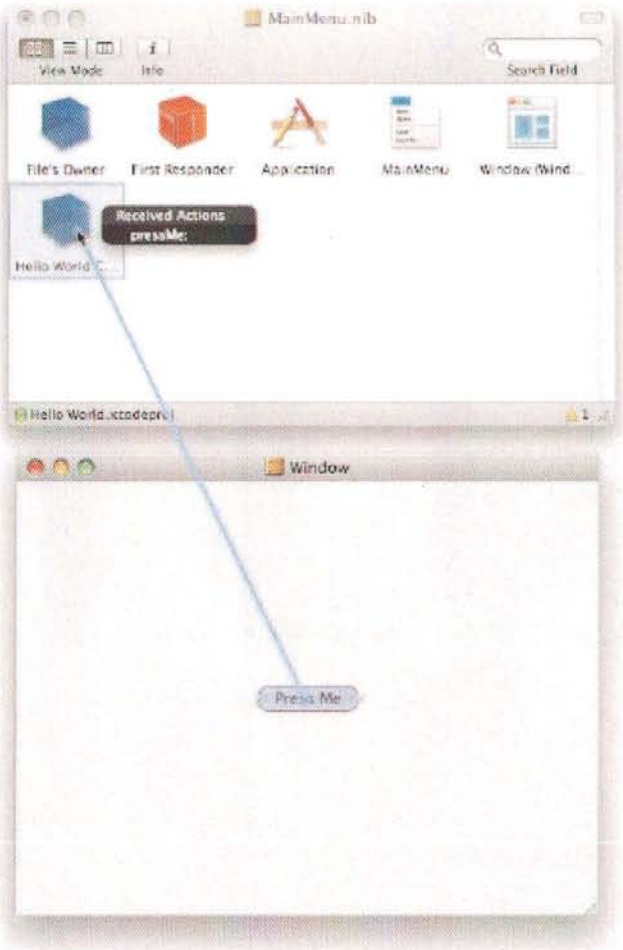


Figure 15: Hooking up a button to an action

That's it! You have now hooked up the button to our action method. Let's go back to Xcode and run our application to see if it worked. After our application launches, click the button a few times and check the console for any output. Xcode 3 no longer shows the Console window by default, but you can open it with the Run > Console command, or Command-Shift-R. If the action was properly hooked up in Interface Builder, you should see output in the console similar to this:

```
2008-02-06 10:43:15.479 Hello World[5864:10b] Button pressed
2008-02-06 10:43:16.079 Hello World[5864:10b] Button pressed
2008-02-06 10:43:19.479 Hello World[5864:10b] Button pressed
```

If you want to always see the Console window like previous versions of Xcode (I do), you can change this in the Xcode preferences. Under the Debugging tab, in the On Start popup, choose Show Console.

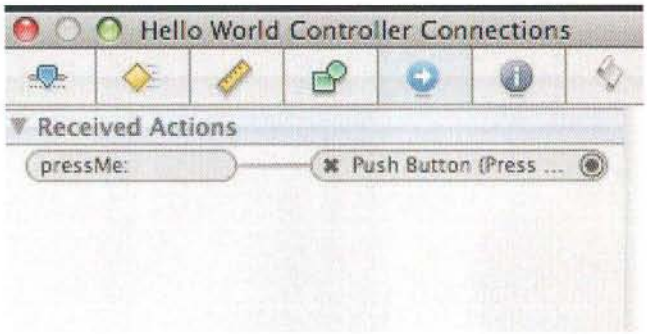


Figure 16: Received actions

Since actions are a two-way relationship, you can also verify this by viewing the connections of the button.

Push buttons are not the only GUI components that send actions. Menu items, checkboxes, radio buttons, and popup buttons all send actions. The procedure for hooking these up is always the same. Create an action method in your code and then control drag in Interface Builder to hook up a component to an action. Now that you know how to do it, I won't go over the detailed procedure in the future.

Outlets

Actions are great, but that's only half the puzzle. We still need to be able to get information out of the GUI. If we have a text field, for example, we will probably want to use what the user entered into it. Let's cover that now.

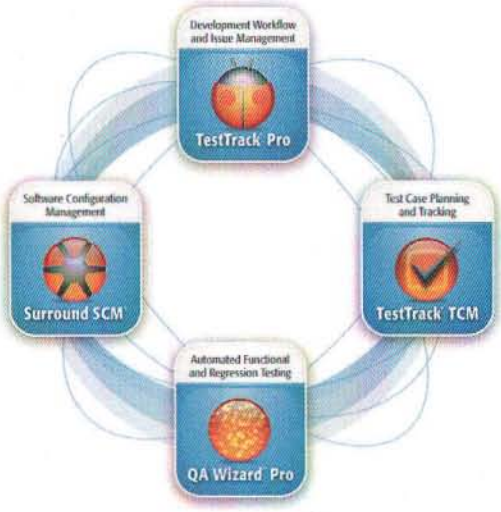
Find the Text Field component in the Library by searching for "text field" and add it to the window by dragging it from the Library, just like we did with the label and button. Be sure not to mistakenly choose the Text Field Cell, which is not the same thing. Place the text field above the button, as shown in Figure 17.

Seapine Software

Satisfy Your Quality Obsession

Software quality and reliability are your lifelines to customer loyalty and profitability. Rigorous quality discipline and seamless collaboration between your development and QA teams will bring your customer-focused strategy to life.

With offices worldwide and over 8,500 customers, Seapine Software is the leading provider of integrated quality-centric application lifecycle management (ALM) solutions. For over 12 years Seapine's easy-to-use, award-winning tools have helped companies achieve success by streamlining communications, improving traceability, and making development and QA teams more productive.



Satisfy Your Quality Obsession.
Visit Seapine Software at **www.seapine.com/qualityready1**

Integrated ALM for a
Quality-Ready Advantage

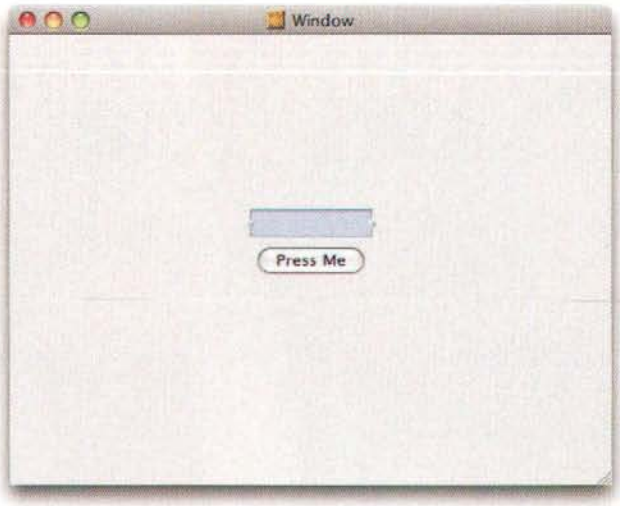


Figure 17: Added text field

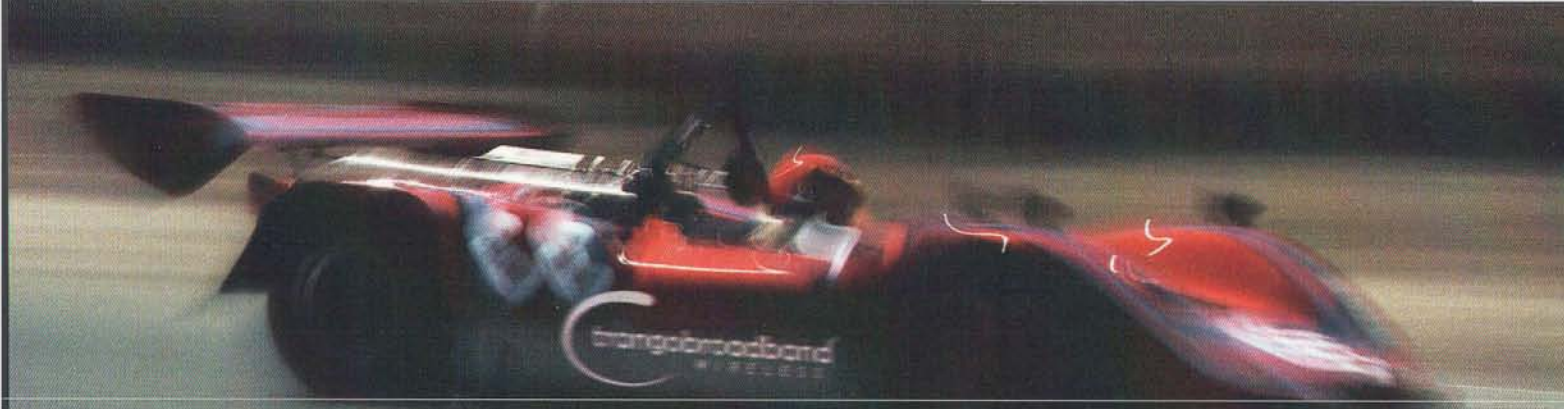
If we now run the application, you will be able to type into the text field, as you might expect. Getting what you type out of the text field requires some code. Back in Xcode, add an instance variable named `_textField` to our class, as shown in Listing 4, and save the file.

Listing 4: HelloWorldController.h with an outlet

```
#import <Cocoa/Cocoa.h>

@interface HelloWorldController : NSObject
{
    IBOutlet NSTextField * _textField;
}

- (IBAction) pressMe: (id) sender;

@end
```

This instance variable is declared like any other instance variable, except it has the `IBOutlet` prefix before the type. Again, this is for integration with Interface Builder and marks this instance variable as an *outlet*. Outlets can be hooked up to GUI components in Interface Builder, and that's the next step.

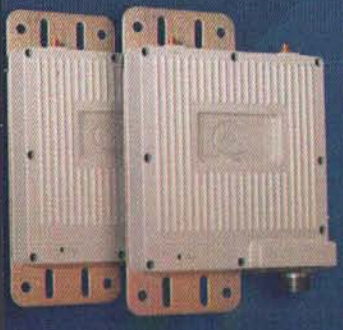
Back in Interface Builder, control drag *from* the Hello World Controller *to* the text field. Again, when you release the button, you should get a popup window allowing you to select `_textField`, as shown in Figure 18. Choose `_textField` to make the connection. You can verify the connection by looking at the Connection tab of the Inspector panel, if you like.



WE OUTPERFORM OUR COMPETITION

Nobody beats us on...

- **Price**
  Save thousands on multi-location T1 services
- **Performance**
  6 to 30× faster than T1 service
- **Reliability**
  Best-in-class fixed wireless solutions



Just one of many wireless solutions from Trango...

ATLAS 5010 Series™
Wireless Ethernet Bridge

- 45 Mbps of sustained throughput
- Capable of 5.3 and 5.8 GHz
- Point-to-point, OFDM
- Up to 40-mile range
- Adaptable Rate Modulation

Wireless Connectivity IS Powered by Trango

Trango Broadband Wireless
A division of Trango Systems, Inc.

www.trangobroadband.com
Phone: +1 (858) 653-3900

trangobroadband WIRELESS

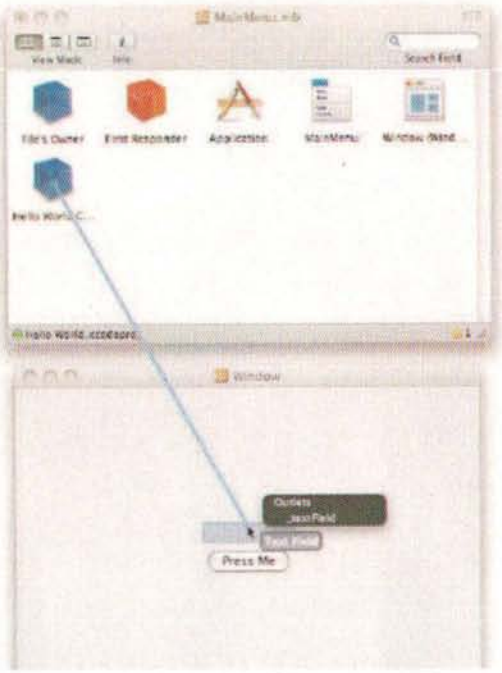


Figure 18: Hooking up an outlet

With the outlet connection made, we can now use our `_textField` instance variable. It will automatically be set at application load time. You'll notice the type of `_textField` is `NSTextField`. This is the Cocoa class for text field controls. It has many methods, but the one we are interested in is `stringValue`. This method, unsurprisingly, returns the contents of the text field as an `NSString`. Modify our action method to log this value when the button is pressed, as shown in Listing 5.

Listing 5: HelloWorldController.m using an outlet

```
#import "HelloWorldController.h"

@implementation HelloWorldController

- (IBAction) pressMe: (id) sender;
{
    NSLog(@"Button pressed");
    NSLog(@"Text field is: %@", [_textField stringValue]);
}

@end
```

Now run the application again. Type some text into the text field, such as "Hello!", and press the button. You should see text in the console window similar to:

```
2008-02-06 11:57:50.697 Hello World[6089:10b] Button pressed
2008-02-06 11:57:50.698 Hello World[6089:10b] Text field is:
Hello!
```

You've just hooked up and used your first outlet! Outlets can be used to hook up any GUI component to your code. They are used quite a bit, as you will see once you start more Cocoa programming. With actions and outlets, we can create a real GUI application.

Rectangle Area Calculation

Let's bring back our old friend from a few articles ago, the Rectangle class. Add a new class named Rectangle to your application, and make sure the header and source files match Listing 6 and Listing 7, respectively.

Listing 6: Rectangle.h

```
#import <Foundation/Foundation.h>

@interface Rectangle : NSObject
{
    float _leftX;
    float _bottomY;
    float _width;
    float _height;
}

@property float leftX;
@property float bottomY;
@property float width;
@property float height;
@property (readonly) float area;
@property (readonly) float perimeter;

- (id)      initWithLeftX: (float) leftX
                  bottomY: (float) bottomY
                   rightX: (float) rightX
                     topY: (float) topY;

@end
```

Listing 7: Rectangle.m

```
#import "Rectangle.h"

@implementation Rectangle

@synthesize leftX = _leftX;
@synthesize bottomY = _bottomY;
@synthesize width = _width;
@synthesize height = _height;

- (id)      initWithLeftX: (float) leftX
                  bottomY: (float) bottomY
                   rightX: (float) rightX
                     topY: (float) topY
{
    self = [super init];
    if (self == nil)
        return nil;

    _leftX = leftX;
    _bottomY = bottomY;
    _width = rightX - leftX;
    _height = topY - bottomY;

    return self;
}

- (float) area
{
    return _width * _height;
}

- (float) perimeter
{
    return (2*_width) + (2*_height);
}

@end
```

This `Rectangle` class is similar to what we've used in previous articles, except that I've updated it with Objective-C 2.0 properties. In the header file, you'll see several `@property` lines. These declare *properties* and must match the following syntax:

```
@property (options) type name;
```

The `type` is any valid Objective-C type, such as C primitives or Objective-C classes. In our case, we're using the `float` primitive type. One of the `options` available is `readonly`. We used this on the `area` and `perimeter` properties. Another option is `readwrite`. This is the default, which is why we don't specify it on the rest of our properties. The `@property` declaration declares accessor methods for us. A `readonly` property declares a getter method only, while a `readwrite` property declares both a getter and a setter. Thus, this single line:

```
@property float leftX;
```

is the same as declaring these two methods:

```
- (float) leftX;
- (void) setLeftX: (float) leftX;
```

There are other options, but we don't need to know about them just yet. It gets even better. Inside the source file, the `@synthesize` keyword generates method implementations for us. Thus, this single line:

```
@synthesize leftX = _leftX;
```

is the same as these eight lines of code:

```
- (float) leftX;
{
    return _leftX;
}

- (void) setLeftX: (float) leftX;
{
    _leftX = leftX;
}
```

As you can see, properties can save us lots of monotonous typing. You do need to match up property names to instance variables, but that's a small price to pay. And as a bonus, if you name your instances variables the same as your properties, you don't need the `"="` part of the `@synthesize` lines. I still prefer underscore prefixes for my instance variables, but that's just personal taste.

You'll notice that we still implement the `area` and `perimeter` methods. This is completely legal, and it shows how you can mix property declarations and custom method implementations, if you need to do so. We need to do it because these methods are calculations and have no corresponding instance variables.

Since properties are Leopard-only, this code will not compile in previous versions of Xcode. If you want to run this on earlier versions of Xcode and Mac OS X, you'll need to replace all the `@property` lines with their corresponding accessor methods.

We're now going to create a GUI around this `Rectangle` class to allow the user to perform area and perimeter calculations. First, set up our user interface in Interface Builder. Drag labels, text fields, and a button into the window, and resize the window so it looks like Figure 19.

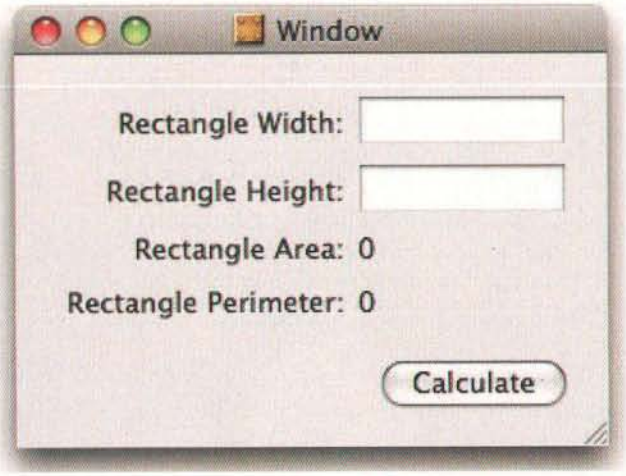


Figure 19: Rectangle window layout

When you're dragging things around and setting up the GUI, you will probably see dashed blue lines pop up, as shown in Figure 20. These are guides that give you hints about Apple's Human Interface Guidelines (HIG). The HIG is a document that specifies many guidelines about designing consistent user interfaces for Mac OS X. It's a big document, and a good read if you plan on doing any serious Cocoa development. Part of the information included in the HIG is spacing guidelines of components, such as how many pixels should be used as border for windows. Luckily, Interface Builder shows us these blue HIG guides so that we don't have to count pixels to make sure we're laying things out correctly.

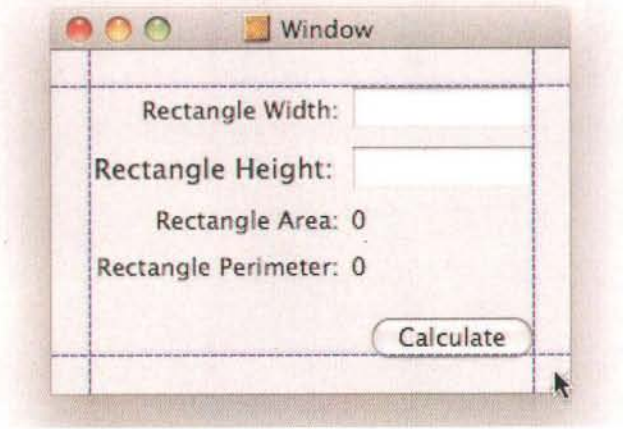


Figure 20: Rectangle window with HIG guides

With our user interface in place, we need to go back into Xcode to hook up these components. Modify the `HelloWorldController.h` header file to match Listing 9.

Listing 9: HelloWorldController.h for rectangles

```
#import <Cocoa/Cocoa.h>

@class Rectangle;

@interface HelloWorldController : NSObject
{
    IBOutlet NSTextField * _widthField;
    IBOutlet NSTextField * _heightField;
    IBOutlet NSTextField * _areaLabel;
    IBOutlet NSTextField * _perimiterLabel;

    Rectangle * _rectangle;
}

- (IBAction) calculate: (id) sender;

@end
```

With these outlets and actions in place, save this file, go back to Interface Builder, and make the appropriate connections. If you did it properly, the connections for Hello World Controller should look like Figure 21.

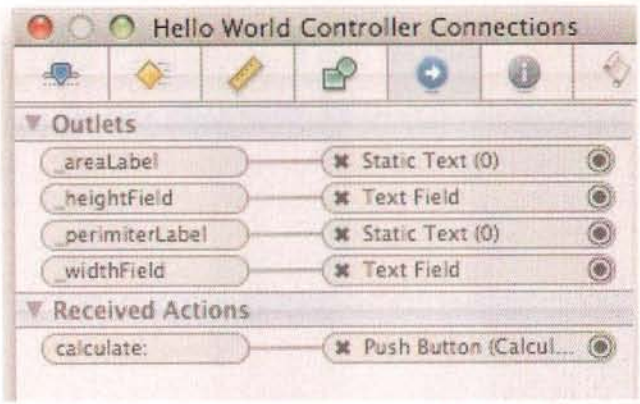


Figure 21: Connections for Hello World Controller

Now, we can finish implementing our controller class. Modify the `HelloWorldController.m` source file to match Listing 10.

Listing 10: HelloWorldController.m for rectangles

```
#import "HelloWorldController.h"
#import "Rectangle.h"

@implementation HelloWorldController

- (id) init
{
    self = [super init];
    if (self == nil)
        return nil;

    _rectangle = [[Rectangle alloc]    initWithLeftX: 0
                                          bottomY: 0
                                           rightX: 5
                                             topY: 10];

    return self;
}

- (void) updateAreaAndPerimeter
{
    [_areaLabel setFloatValue: _rectangle.area];
    [_perimiterLabel setFloatValue: _rectangle.perimeter];
}
```

```
- (IBAction) calculate: (id) sender
{
    _rectangle.width = [_widthField floatValue];
    _rectangle.height = [_heightField floatValue];
    [self updateAreaAndPerimeter];
}

- (void) awakeFromNib
{
    [_widthField setFloatValue: _rectangle.width];
    [_heightField setFloatValue: _rectangle.height];
    [self updateAreaAndPerimeter];
}

@end
```

We've had to beef this up a bit, but it's still pretty simple. In our constructor, we create a new `Rectangle` instance with a width of 5 and a height of 10. Our `calculate:` action sets the rectangle's width and height from the text fields, using the `floatValue` method. It then calls `updateAreaAndPerimiter` to set the area and perimeter labels according to the rectangle's current area and perimeter. This code also demonstrates another benefit of properties: you can use the *dot notation* to access them. Thus, instead of calling setter methods:

```
- (IBAction) calculate: (id) sender
{
    [_rectangle setWidth: [_widthField floatValue]];
    [_rectangle setHeight: [_heightField floatValue]];
    [self updateAreaAndPerimeter];
}
```

we can use the dot notation, `_rectangle.width =`, as shown in Listing 10.

The only other new bit is the `awakeFromNib` method. This is called automatically by Cocoa at application startup. We use this to set up the initial values of the text fields and labels when our application is started. You'd think we could do this in our constructor, but there's a bit of a problem with that.

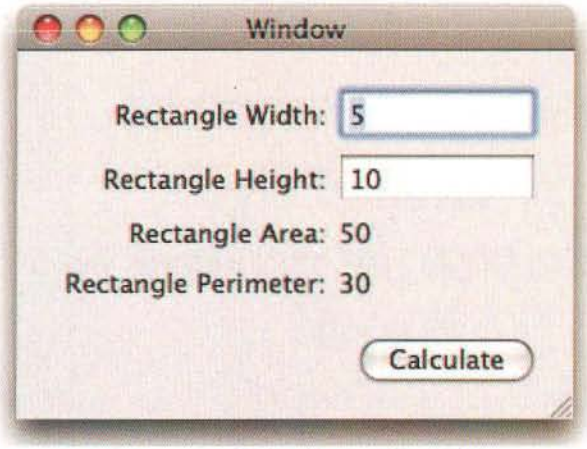


Figure 22: Initial Hello World window

Remember that objects inside a nib file are "freeze-dried" or "put to sleep" when Interface Builder saves the nib file. When you application runs, these saved objects are

"reconstituted" or "awoken" at application launch time. The problem is that we don't know what order these objects are awoken and created. Thus, the text fields and labels may not be initialized when our constructor is called. To get around this situation, Cocoa first instantiates *all* sleeping objects in the nib, and then calls `awakeFromNib` on each of these objects. You don't *have* to implement `awakeFromNib`, but we take this opportunity to setup the initial GUI state. As a rule, if you need to work with outlets, you must not do so in the constructor. Use the `awakeFomNib` method. Thus, when our application is first run, you will see a window that looks like Figure 22 (above).

We can now change the width and height and click the Calculate button. If everything went as planned, the area and perimeter will be updated as shown in Figure 23.

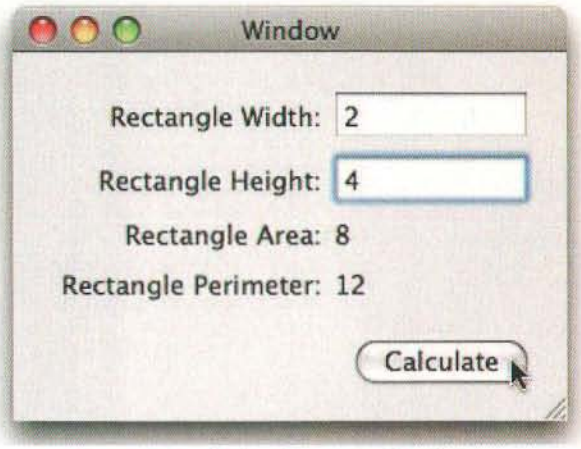


Figure 23: Calculated area and perimeter

Conclusion

Well, how about that! You've created your first honest-to-goodness Mac OS X GUI application. As you can see, it often does not take that much code to produce nice looking user interfaces. Interface Builder is the key to this ease. Now that you've got the basics of Interface Builder and GUI Mac OS X applications down, we can move on to bigger and better topics. The full source to the project will be available for download from the MacTech website.

About The Author

Dave Dribin has been writing professional software for over eleven years. After five years programming embedded C in the telecom industry and a brief stint riding the Internet bubble, he decided to venture out on his own. Since 2001, he has been providing independent consulting services, and in 2006, he founded Bit Maki, Inc. Find out more at <http://www.bitmaki.com/> ***and*** <http://www.dribin.org/dave/>.

Advertiser/Product Index

Absolute Software - LoJack 85
ActiveState Software Inc. 19
Addlogix (formerly CompuCable Mfg. Group) 52
Akor 26
Aladdin Knowledge Systems, Inc. 51
Allume Systems, Inc. 25
Ambrosia Software Inc. 23
Aqua Connect, Inc. 41
Berkeley Varitronics Systems, Inc. 49
Brad Sniderman 83
CalDigit 9
Cynergy Data 42
Edgeos, Inc. 37
eSellerate/MindVision 45
Faronics Corporation 55
Fontlab Ltd. 4
Freeridecoding 30
Future Media Concepts 10
Hewlett-Packard Company ..... IFC
IGC, Inc. / MaxEMail.com 62
IGG Software, LLC 13
iSkin Inc. 30
Kerio Technologies Inc. 14
LANrev LP 69
LassoSoft LLC 12
Lemke Software GmbH 37
LithiumCorp 29
Mac Rentals, Inc. 65
MacForge.net 80
MacMall 2-3
MacResource Computers & Service 73
MacSpeech, Inc. 16
MacTech Domains 86
MacTech Magazine 77
MacTek 31
Mark/Space Inc. 75
MARWARE 22
MDG Computer Services, Inc. 40
Microsoft ..... BC
Mushkin 1
MYOB US, Inc. 21
Now Software 71
OlympicControls Corp. 43
Other World Computing 11
Other World Computing 47
Ovolab 48
Parallels Inc. 61
Peachpit Press 17
Powerbookmedic.com 67
ProjectWizards 46
RAMJET 31
REAL Software, Inc. 15
Sans Digital 57
Seapine Software, Inc. 79
Small Dog Electronics ..... IBC
Small Tree Communications 27
Stellar Information Systems Ltd. 50
SupaCam / Nisis 63
TechRestore 24
Toon Boom Animation 35
Trango Broadband Wireless 81
USGlobalSat, Inc 59
Utilities4Less.com 72
Walter Karl, An InfoUSA Company 34
WIBU-SYSTEMS AG 33
WorldSync, Inc. 53
Yazsoft.com 64
ZAGG Inc (dba ShieldZone) 66

AccountEdge • MYOB US, Inc. 21
Aqua Connect Terminal Server • Aqua Connect, Inc. 41
BookEndz • OlympicControls Corp. 43
CodeMeter • WIBU-SYSTEMS AG 33
DAS, NAS, and SAN Storage Solutions • Sans Digital 57
Deep Freeze • Faronics Corporation 55
Domain Registration • MacTech Domains 86
Edge-corE Switches • Small Tree Communications 27
eMail Appending • Walter Karl, An InfoUSA Company 34
eSellerate • eSellerate/MindVision 45
eTailer • MDG Computer Services, Inc. 40
Font Editor • Fontlab Ltd. 4
Gateway • Cynergy Data 42
GPS System • USGlobalSat, Inc 59
Graphic Converter • Lemke Software GmbH 37
HASP • Aladdin Knowledge Systems, Inc. 51
iBank/iBiz • IGG Software, LLC 13
invisibleSHIELD by ZAGG • ZAGG Inc (dba ShieldZone) 66
iSkin • iSkin Inc. 30
IT Training • Future Media Concepts 10
Kerio Server Software • Kerio Technologies Inc. 14
Komodo • ActiveState Software Inc. 19
KVM Switch • Addlogix (formerly CompuCable Mfg. Group) 52
LANrev • LANrev LP 69
Lasso • LassoSoft LLC 12
Law Offices • Brad Sniderman 83
Lithium Network Monitoring • LithiumCorp 29
LoJack for Laptops • Absolute Software - LoJack 85
Long Distance Phone Service • Utilities4Less.com 72
MacMall • MacMall 2-3
MacResource Computers • MacResource Computers & Service 73
MacSpeech Dictate • MacSpeech, Inc. 16
MacTech Magazine • MacTech Magazine 77
maxemail.com • IGC, Inc. / MaxEMail.com 62
Memory • Mushkin 1
Memory • RAMJET 31
Memory Products • Other World Computing 11
Mercury Storage Solutions • Other World Computing 47
Merlin • ProjectWizards 46
MiniMax Keyboard • Akor 26
Missing Sync • Mark/Space Inc. 75
Now Up-to-Date • Now Software 71
Office 2008 for Mac • Microsoft ..... BC
Open Source Directory • MacForge.net 80
Parallels Desktop for Mac • Parallels Inc. 61
Peachpit Press • Peachpit Press 17
Phlink • Ovolab 48
Powerbookmedic.com • Powerbookmedic.com 67
Printers • Hewlett-Packard Company ..... IFC
Project X • MARWARE 22
Raid Storage • CalDigit 9
REALbasic • REAL Software, Inc. 15
Recovery Utilities • Stellar Information Systems Ltd. 50
Rentals, Computer • Mac Rentals, Inc. 65
Repairs and Upgrades • TechRestore 24
Security Services • Edgeos, Inc. 37
SmallDog.com • Small Dog Electronics ..... IBC
SmartBackup • Freeridecoding 30
Spectrum Analyzer • Berkeley Varitronics Systems, Inc. 49
Speed Download • Yazsoft.com 64
StuffIt • Allume Systems, Inc. 25
SupaCam • SupaCam / Nisis 63
SyncDek • WorldSync, Inc. 53
TestTrack • Seapine Software, Inc. 79
Toon Boom Studio • Toon Boom Animation 35
Training • MacTek 31
Trango Broadband • Trango Broadband Wireless 81
WireTap Studio • Ambrosia Software Inc. 23

THE MACTECH SPOTLIGHT

Eberhard Rensch

Founder, Pleasant Software

http://www.pleasantsoftware.com

What do you do?

I'm the founder and main developer of Pleasant Software.

Pleasant Software has been developing and selling high quality software products for the Macintosh since 1989. Currently, our three main products are "Übercaster" which is the most advanced podcast production suite for the Macintosh, "ShowMacster" which revolutionizes video conferencing by seamlessly extending Apples iChat application and "PiP", a free tool to present video from any webcam on the desktop.

How long have you been doing what you do?

First programming in 1982 on an Apple II in school, developing Mac software since 1988 starting with MPW on a Mac SE.

What was your first computer:

EACA VideoGenie I (a TRS-80 clone)

Are you Mac-only, or a multi-platform person?

All current main projects are Mac only Obj-C/Cocoa applications. There's also some software I've written in pure Java (with Swing GUI) a while ago, which I still support (running on Mac & Windows). I also used to work as freelance developer and consultant, which was all Windows stuff.

What attracts you to working on the Mac?

The Cocoa framework and Objective-C. I also personally have used Macs for more than 19 years and I love Mac OS X...

What's the coolest thing about the Mac?

From a developer's perspective: Cocoa! I've worked with a lot of frameworks during the last 2 decades and Cocoa is clearly one of my absolute favorites. It's easy to learn and extremely powerful! It saves a lot of work, especially for small companies like mine. I'm pretty sure that I couldn't have realized a project like Übercaster without Cocoa.

If I could change one thing about Apple/OS X, I'd:

I'd get rid of closed systems policy like .Mac-only functions in iSync or the missing/closed SDKs for AppleTV and iPod (classic/nano). The announcement of an iPhone / iPod Touch SDK is definitely a step in the right direction. We'll see how open this SDK will be...

What's the coolest tech thing you've done using OS X?

Back in 2001 I bought a PowerBook G4/400 Titanium with Mac OS X 10.1 and moved into the Alpujaras/Spain for 3 months to write a quite complicated and sophisticated business solution program. So I did, in a "Cortijo" on the top of a mountain, without access to the Internet or AC power (only one solar panel with an AC converter). That application was a true "greener Apple" project. And it was a really great experience!

The next way I'm going to impact IT/OS X/the Mac universe is:

To quote Apple: We do not comment on future products :) But Leopard introduced a lot of great technologies and I intend to use them...

If you or someone you know belongs in the MacTech Spotlight, let us know! Send details to editorial@mactech.com

stocked, savvy *and* socially responsible.

The latest and greatest, with extras

At Small Dog Electronics, we carry the newest Mac stuff, as well as the older goodies you're looking for. Often when Apple announces a new product, we have access to older models, many with a significant price drop. Check our new website for both new toys and tried & true favorites.

Friendly, knowledgeable staff

We know Macs, and we know them well. If there's a question we can't answer, we'll find it for you. Every member of our staff (even the ones not in Sales) is a certified Apple Product Professional, and over three-quarters of us have also received our Apple Sales Professional status.

Earth Day, every day

We pride ourselves on being a business with multiple bottom lines, which includes an active commitment to social responsibility and environmental initiatives. We offer proper disposal of electronic waste for our customers, and this year, we're hosting our 2nd Annual Free eWaste Recycling Event on April 19th at our S. Burlington location. But, even if you don't live in the area, when you purchase from Small Dog, you're helping to support a greener environment! Read more about our ewaste recycling program and annual event at Smalldog.com.

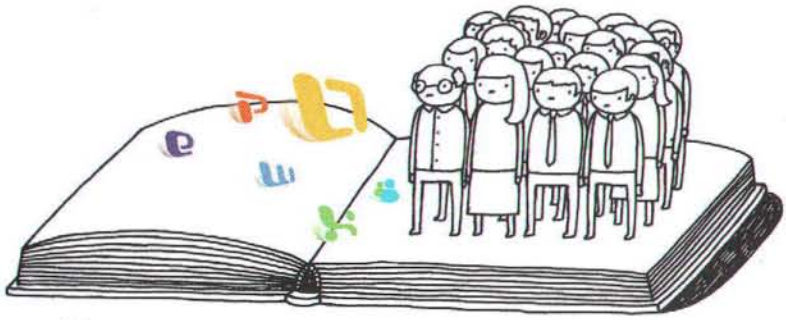
New Website!

Small Dog Electronics

Always By Your Side

- over 3,000 products
- 5-star online merchant rating

www.smalldog.com **800-511-MACS** Apple Specialist



Everyone on the same page

Work together. Different machines? Different platforms? No matter. You can all speak the same language.

SimplifyYourWork2008.com

Microsoft® Office:mac 2008